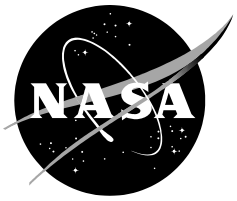
NASA/TM–20250005344



# Flight Modeling and Trajectory Prediction for Diverse Applications

*Michael Abramson*
*Senior Engineer, Crown Innovations, Inc., Moffett Field, California, 94035-1000*

*Scott Sahlman*
*Senior Software Engineer, Universities Space Research Association, Moffett Field, California, 94035-1000*

May 2025

# NASA STI Program Report Series

The NASA STI Program collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

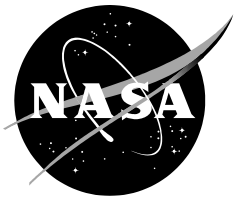
For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- Help desk contact information:

https://www.sti.nasa.gov/sti-contact-form/ and select the "General" help request type.

NASA/TM–20250005344



# Flight Modeling and Trajectory Prediction for Diverse Applications

*Michael Abramson*
*Senior Engineer, Crown Innovations, Inc., Moffett Field, California, 94035-1000*

*Scott Sahlman*
*Senior Software Engineer, Universities Space Research Association, Moffett Field, California, 94035-1000*

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California, 94035-1000

May 2025

## Acknowledgments

The authors want to express their gratitude to Confesor Santiago, Mohamad Refai, Eric Wahl, and James Phillips for their support, encouragement, and contributions in development of software tools used in this paper, and to Min Xue for providing $Fe^3$ data used in validation of MATG.

## Abstract

A fast and flexible Multimodal Adaptable Trajectory Generator (MATG) was developed and tested on the Java Architecture for Detect-and-Avoid (DAA) Extensibility and Modeling (JADEM) software research platform. MATG can generate trajectories for various pilot procedures and operations, including traditional point-to-point flights in the National Airspace System (NAS), maneuvers required for piloted and autonomous detect-and-avoid (DAA) systems, and new flight procedures for Urban Air Mobility (UAM) airspace. MATG can handle various constraints in many different combinations. It can be easily adapted for modeling traditional aircraft using aircraft performance models from the Base of Aircraft Data (BADA), and unconventional aircraft not currently supported by BADA, such as electric Vertical Take-Off and Landing (eVTOL) aircraft relevant for UAM operations. This paper describes the MATG data model, algorithm, and applications.

# Table of Contents

## Table of Tables



## Table of Figures

## Nomenclature

| | |
|---|---|
| AAM | Advanced Air Mobility |
| ADS-B | Automatic Dependent Surveillance – Broadcast |
| AGL | above ground level |
| AIDL | Aircraft Intent Description Language |
| APM | Aircraft Performance Model |
| BADA | Base of Aircraft Data |
| CAS | calibrated airspeed |
| CSV | comma-separated values |
| DAA | detect-and-avoid |
| DOF | degree-of-freedom |
| DTV | Dynamic Traffic Viewer |
| ETAG | Estimated Times of Arrival (ETAs) Generation service |
| eVTOL | electric Vertical Take-Off and Landing |
| FCDL | Flight Commands Definition Language |
| FIDL | Flight Intent Description Language |
| FPA | Flight Path Angle |
| JADEM | Java Architecture for DAA Extensibility and Modeling |
| MATED | Metric Analysis Tool for Encounter Data |
| MATG | Multimodal Adaptable Trajectory Generator |
| NAS | National Airspace System |
| IFR | instrumental flight rules |
| TAS | true airspeed |
| TCP | Trajectory Change Point |
| TCM | Tactical Conflict Management |
| UAM | Urban Air Mobility |
| UAS | unmanned aircraft systems |
| VFR | visual flight rules |

# 1 Introduction

There is an increasing demand for integrating new vehicle types, innovative aviation technologies, and diverse operations into the National Airspace System (NAS), while also enabling the large-scale use of electric Vertical Take-Off and Landing (eVTOL) aircraft for novel Urban Air Mobility (UAM) applications. An emerging concept of UAM, which is a part of a broader concept known as Advanced Air Mobility (AAM), aims to expand transportation networks to include short-range flights that rapidly transport people and goods, at low altitudes in and around growing metropolitan areas. The UAM concept can provide a fast and practical mobility alternative to ground transportation for the general public that can eventually become cost-effective. Successful implementation of UAM operations is predicated upon maintaining or exceeding the level of safety and performance achieved by current operations in the NAS. Achieving these goals requires extensive research efforts that present unique challenges for modeling/simulation software tools. High traffic density and operational tempo, unusual airspace structures, and different performance characteristics of eVTOL aircraft are just a few of many such challenges. One promising simulation tool that can meet them is the Java Architecture for detect and avoid (DAA) Extensibility and Modeling (JADEM), previously developed to support research on integration of unmanned aircraft systems (UAS) into the NAS [4]. JADEM was already used as a fast-time simulation tool to research alerting and guidance functions of UAM aircraft [17,23]. These simulations confirmed the ability of JADEM to model high-density UAM traffic by orders of magnitude faster than real time.

One of the keys to JADEM's versatility is the use of a flexible flight data model for all flight modeling and trajectory generation tasks. JADEM uses this model to define a *TrajectoryPredictor* interface and provides a fast kinematic Multimodal Adaptable Trajectory Generator (MATG) as a default implementation of this interface. Besides the use in JADEM, MATG can be used as a standalone trajectory generator and as a library that can be called by other tools and services. For example, MATG is used by the Estimated Times of Arrival (ETAs) Generation (ETAG) service [16].

The structure of this paper is organized as follows. Section 2 reviews some background on flight modeling and trajectory generation systems and algorithms. Section 3 provides a high-level overview of the JADEM simulation tool. Section 4 describes the underlying flight data model and MATG logic and algorithm. Section 5 describes implementation of this algorithm and MATG performance. Section 6 describes methods and tools developed for common flight modeling and data analysis tasks and used to validate MATG. Section 7 describes MATG Verification & Validation with examples for different applications. Section 8 includes a summary and suggestions for future MATG improvements and extensions.

# 2 Background

Aircraft trajectory generation requires using diverse sources of information, including flight plans, flight procedures, airspace constraints, weather forecasts, aircraft performance data, airlines' preferred speeds, and the aircraft state. Different air traffic simulation systems and tools, such as NASA's Center-TRACON Automation System (CTAS) [11], Future ATM Concepts Evaluation Tool (FACET) [6], Airspace Concept Evaluation System (ACES) [19], Flexible engine for Fast-time evaluation of Flight environments ($Fe^3$) [24], and trajectory generators of various fidelity [8–10,15,22,25], differ in how they use this information and how they balance conflicting requirements of accuracy, realism, flexibility, practicality, and performance.

Many trajectory generation systems rely, to varying degrees, on Eurocontrol's Base of Aircraft Data (BADA) [2]. CTAS Trajectory Synthesizer (CTAS TS) [15, 22] originally included its own aircraft performance models

(APM) for several most-widely-used air frames, but it was extended to support BADA APM as well [3]. This allowed using the extended CTAS/BADA TS for modeling all aircraft types supported in BADA. BADA data was used to govern the aircraft performance in Kinematic Trajectory Generator (KTG) [25] and for fuel burn estimation in [8].

However, BADA has its limitations. It was developed for traditional airspace operations, mission profiles, and mostly for winged aircraft types. The new UAM/AAM operations and mission profiles [20] rely on capabilities of novel and rapidly developed eVTOL aircraft that don't have adequate BADA models. The higher fidelity six-degree-of-freedom (6DOF) models have been proposed for quadrotor aircraft [7,13]. These models, however, are less practical since their parameters, such as the moment of inertia tensor, are usually not shared by manufacturers. Therefore, even if simulation tools, such as Fe$^3$ [24], include the higher fidelity APM for non-traditional aircraft, using such APM doesn't guarantee the higher accuracy of generated trajectories because of necessary approximations and assumptions for unknown parameters.

From a practicality standpoint, the 4DOF kinematic trajectory generators such as KTG [25] offered an attractive solution. Unfortunately, KTG was available only in Matlab or as a standalone Java application. It was also integrated in ACES, but its integration in other simulation tools would require a nontrivial effort.

Therefore, there was a need for a trajectory generator that could model a mix of aircraft types that could be very diverse, including traditional airliners, general aviation aircraft, unmanned aircraft, and winged and eVTOL aircraft. At the same time, a trajectory generator should be sufficiently flexible for modeling new UAS and UAM/AAM operations so that it can be used to generate point-to-point trajectories, dead-reckoned trajectories based on known aircraft state, partially-updated trajectories for avoidance maneuvers, and trajectories for complex flight patterns or loitering flights. Finally, modeling DAA systems for multiple flights involves generating a large number of trial trajectories that imposes strict requirements on trajectory generator performance.

These requirements justified development of the new Multimodal Adaptable Trajectory Generator (MATG). MATG was originally created and used to support all flight physics modeling needs in JADEM, which is described in the next section.

# 3 JADEM Overview

The Java Architecture for Detect-and-Avoid Extensibility and Modeling (JADEM) was originally developed as a general-purpose simulation tool for evaluating DAA concepts and their safety characteristics for integrating unmanned aircraft into the NAS [4]. JADEM was built to support NAS-wide assessments and parametric trade-space studies, but also can be used for fast closed-loop simulations of UAM operations, as well as for human-in-the-loop studies and for flight tests that involve real aircraft. JADEM provides a flexible and extensible software platform that includes models and algorithms for evaluating all major conflict alerting and guidance functions as shown in Figure 1.

JADEM includes Flight dynamics, Detect and Track, Alerting and Guidance, Pilot, Navigation, and Surveillance Models such as ADS-B and onboard radar. The models are managed by a driver called SaaControl, which provides all required simulation functionality. The name "SaaControl" is derived from Sense-and-Avoid (SAA), a legacy term for DAA, which is still used in Europe. SaaControl includes a computationally effective logic to evaluate whether potential conflicts with intruders are close enough to require an avoidance maneuver.

JADEM uses MATG to provide flight physics modeling capability, which relies on a kinematic vehicle model, including bank angle or turn rate, horizontal and vertical accelerations, and altitude-dependent rates of climb

and descent. These parameters can be derived from real aircraft behavior inferred from track data or from APMs defined by users or provided by BADA.

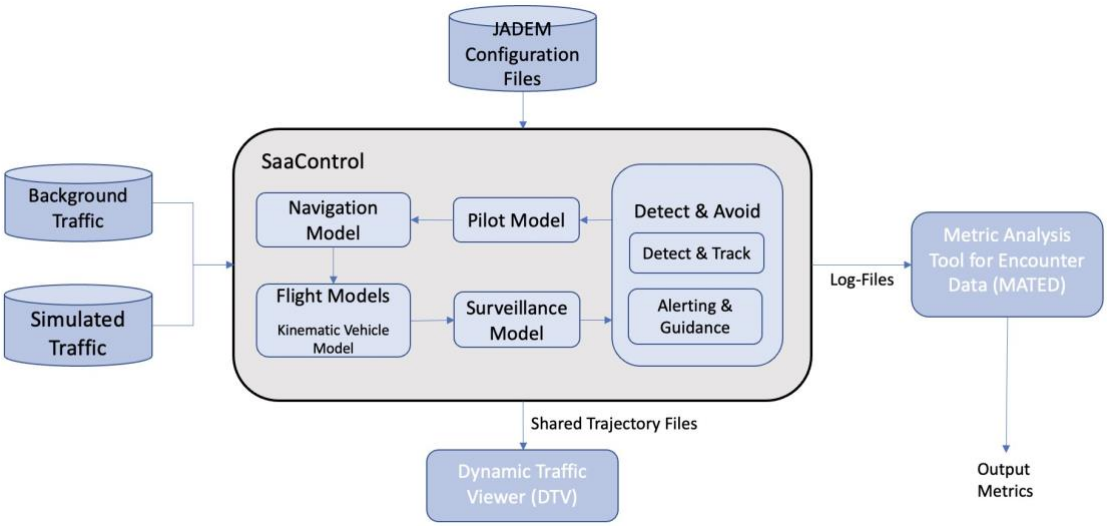



*Figure 1: JADEM Architecture*

In addition to SaaControl, the JADEM ecosystem includes postprocessing / data analysis and visualization tools. One of these tools is the Metric Analysis Tool for Encounter Data (MATED), which generates the final safety and operational suitability metrics. Another useful tool is the Dynamic Traffic Viewer (DTV), which allows the visualization and animation of flight data from trajectory files generated by SaaControl (see figures in Section 7).

Accordingly, JADEM/SaaControl generates two kinds of output:

1. log-files in CSV-format, used by MATED for postprocessing and occasionally useful for researchers.
2. trajectory files for traffic snapshots and maneuvers, viewable in DTV.

# 4 Flight Data and Trajectory Generation

## 4.1 Flight Data

JADEM uses a common "language" for all components dealing with flight input and output, preprocessing, and trajectory generation tasks. In this and subsequent sections all language constructs will be identified using italics. As a language construct, *Flight Data* can be described as the following.

- *AC ID* - an aircraft identifier that uniquely identifies the flight (the only required text element).
- *AC TYPE* - an aircraft type identifier that can be used to load the appropriate Aircraft Performance Model (APM).
- *constraints* - the flight plan (a part of operational intent) as a sequence of aircraft states (the *AC State* objects described below) for waypoints or other constraints, such as speed and altitude restrictions.
- *trajectory* - a sequence of *AC State*s for actual, predicted, or simulated trajectory.

- *commands* - the list of *Flight Command*s for requirements or rules describing a part of aircraft intent that can't be expressed in the language of constraints (see subsection 4.2).
- *control* - the *Flight Control* settings instructing a trajectory predictor how to model the flight.
- *departure* - the *AC State* for the origin of the flight.
- *destination* - the *AC State* for the destination of the flight.
- *parameters* - the *Flight Parameters* object for flight- and flight-phase-specific aircraft performance and operational parameters and limits. These parameters can include accelerations, bank angle or turn rate, and rates of climb and descent that can be altitude-specific.
- *properties* - a map of any other arbitrary properties for this flight.

The *AC State* can include any combination of parameters describing aircraft state, such as:

- *name* (e.g., the name of waypoint).
- *time*.
- *xy* - position of this state in latitude/longitude or Cartesian (x, y) coordinates.
- *altitude* - MSL pressure altitude.
- *ground speed*.
- *tas* - true airspeed (TAS).
- *cas* - calibrated airspeed (CAS).
- *mach* - Mach number.
- *true course*.
- *true heading*.
- *bank angle*.
- *vertical speed*.
- *wind* - wind velocity vector.

An important special case of *trajectory* is an *initial trajectory* including only one initial *AC State* that should always be known (in addition to the flight plan specified by *constraints*) in order to generate a trajectory.

Note that both constraints and trajectory states are represented by *AC State* objects. This is possible because almost all variables in *AC State* can represent a desired, actual, or simulated aircraft state, and all these variables are optional. This allows the use of *constraints* as a flexible representation of flight plan (intent) that

combines the lateral and longitudinal specification of the trajectory. This can be contrasted with the Generalized Profile Interface that supported only the longitudinal specification [15].

## 4.2 Flight Commands Definition Language

Many requirements or rules cannot be expressed in the language of *constraints* because they dictate what should happen for multiple constraints, or regardless of any constraints, or because they aim to redefine behavior specified by constraints. Examples of such requirements might be to maintain temporary speed or altitude, to reduce speed in sharp turns or when flying in certain airspace or below a certain altitude, or to keep speeds or altitudes within predefined limits. Such requirements are considered "commands" that could be used in two ways. They could be executed as a preprocessing step for creating the fully specified list of constraints to be passed to the trajectory generator. Alternatively, the *commands* could be executed by a trajectory generator, such as MATG, which has full access to all flight data, including trajectory state at any given time.

The *commands* are formally specified by the Flight Commands Definition Language (FCDL). FCDL bears some similarities with Flight Intent Description Language (FIDL), which is the highest level in the hierarchy of trajectory specification languages proposed in [5] and ultimately based on Aircraft Intent Description Language (AIDL) [18]. However, FCDL is designed to be simpler and easier to use while allowing sufficient flexibility in specifying conditions of applicability of *commands*. Furthermore, FCDL is a prescriptive rather than a descriptive language. It defines desired (commanded) modifications to aircraft behavior on top of "default" behavior specified by flight plans. Therefore, FCDL shall not be construed as a subset or a superset of FIDL. It is a different language with its own grammar and semantics described below.

A central concept of FCDL is a *Flight Command* that includes the following elements:

- *Name* that doesn't need to be unique, but it must clearly describe the purpose of this Command.
- *Conditions* that define applicability of this *Flight Command*.
- *Actions* that must be executed if all *Conditions* are met (valid).

Each of *Conditions* can include:

- *CHECK* condition at CURRENT (default), PREVIOUS, or NEXT target.
- *Condition Operation* that specifies the relationship between *Variable* and *Value* or *Value*s in range defined in *Command Variable*, such as IS (equals), LT (less than), GT (greater than), IN (in range between *Value* and *Max Value*), and negations: IS NOT equal and NOT IN range.
- *Command Variable* as defined below.

Each of *Actions* can include:

- *ACTION*, such as USE (meaning to use *Value* as defined in *Command Variable*).
- *Action Operation*, such as IS (the only operation that makes sense for USE action).
- *Command Variable* as defined below.

*Command Variable* is the main part of any of *Conditions* or *Actions* that can include the following:

- *Variable*, which can be one of the following: any property of the current aircraft state or a target (next constraint); a derived property, such as heading change or above ground level (AGL) altitude; a flight state or control parameter (e.g., flight phase, airspace type, etc.).
- *Value* that can be set to or compared against the *Variable*. It can be defined as a number, text, or another *Variable*, which can be dynamically evaluated during execution of command.
- *Max Value* that can optionally define the range of values between *Value* and *Max Value* (in this case *Value* plays the role of low bound in this range).
- *Units* - optional units of *Variable* and *Value*.

Thus, the *commands* are defined simply as a list of sequentially executed *Flight Command*s.

The FCDL is representation-agnostic. The same commands can be expressed in abbreviated symbolic notation, in plain English, in JavaScript Objective Notation (JSON), in formal API specs, or in a tabular format that can be used for documenting requirements in FCDL and in input CSV-files for software tools supporting FCDL (e.g., a file for flight commands as an additional input for tools using MATG). In any case, the definition of *commands* is expected to be unambiguous and self-explanatory. Example of a simple command in plain English:

**Maintain cruise speed and altitude en route**

| | |
|---|---|
| *Conditions* | Flight Stage = Enroute |
| *Actions* | USE State TAS = cruise speed knots |
| | USE State Altitude = cruise altitude feet |

This command uses two "dynamic variables" in its *Actions*. Their specific values will be determined from the "cruise speed" and "cruise altitude" *properties* of each flight.

Another example of a command that can be executed in preprocessing:

**Reduce speed before sharp turns**

| | |
|---|---|
| *Conditions* | NEXT Target Course Change > 45 degrees |
| | Target TAS > 70 knots |
| *Actions* | USE Target TAS = 70 knots |

This command will reduce TAS for every constraint with TAS above 70 knots before the next constraint (target) with course change exceeding 45 degrees (a sharp turn) to 70 knots.

### 4.3 Multi-Modal Adaptable Trajectory Generator (MATG)

As outlined in Section 2, the following requirements for MATG are considered most important:

1. good performance,
2. robustness,
3. flexibility.

The first requirement is achieved by using a kinematic solution and minimizing unnecessary calculations. For instance, MATG consistently uses Cartesian coordinates to avoid costly conversions between (*x,y*) and latitude/longitude, and it recalculates airspeeds only when the speed or altitude has changed.

The second requirement means that MATG should never fail and be able to generate a safe and reasonable trajectory for all possible conditions. This is critical because MATG may be called thousands of times during a simulation, and any failures could make the whole simulation futile. It is considered a caller's responsibility to

determine if something unexpected happens, and MATG provides enough information for detecting any abnormal conditions.

MATG's flexibility is achieved by supporting *Flight Data* as a general way to specify its input, output, and control parameters. For all possible uses, MATG accepts an *initial flight* with an *initial trajectory* as its input and returns the *Flight Data* for a *computed flight* with calculated *trajectory* and *updated constraints*.
MATG is typically used to create a trajectory from given *constraints* (such as a flight plan) and an *initial trajectory*, which consists of a single *initial state*. It can also be used to recalculate (update) a part of the *trajectory* in *updated flight* starting at a specified trajectory state corresponding to *initial state*.

### 4.3.1 Algorithm

MATG algorithm is based on linearized numerical equations for aircraft position and velocity vectors, assuming constant velocity for calculating position and constant acceleration for calculating velocity within a time step $\Delta t$. This Euler method is simple, fast, and unconditionally stable. It provides the first order approximation for position and velocity, which is acceptable because:

1. typical time steps are small (1 to 10 sec);
2. a tight control keeps errors in check;
3. integration errors are usually negligible in comparison with various modeling errors (aircraft performance, intent, weather forecast, etc.).

The control loop for MATG kinematic solution is shown in Figure 2.

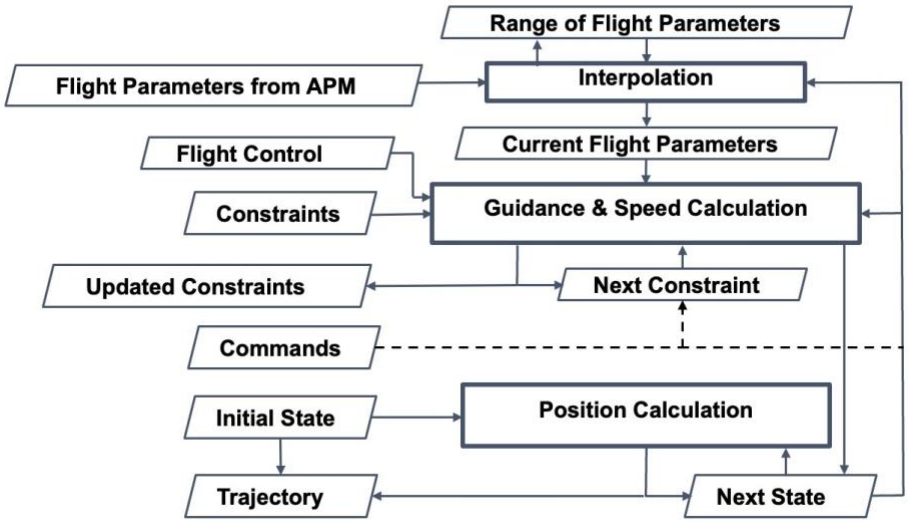



*Figure 2: Control loop for MATG kinematic solution*

The algorithm is built around position calculation using simple extrapolation at given time $t$:

$$x(t + \Delta t) = x(t) + u(t)\sin\theta(t)\Delta t \quad (1)$$

$$y(t + \Delta t) = y(t) + u(t)\cos\theta(t)\Delta t \quad (2)$$

$$z(t + \Delta t) = z(t) + v(t)\Delta t \quad (3)$$

where ($x$,$y$) are Cartesian coordinates of aircraft position $xy$ with $x$-axis pointing to the East and $y$-axis pointing to the North, $z$ is *altitude*, $u$ is *ground speed*, $\theta$ is *true course* measured from South-to-North direction, and $v$ is *vertical speed*.

Without *constraints* the equations (1-3) would generate a straight dead-reckoned trajectory. MATG processes (with some exceptions) one constraint at a time, denoted as *Next Constraint* in Figure 2, which is used to update $\theta$ (see subsection 4.3.3), and velocity vector ($u$,$v$) (see subsections 4.3.4 through 4.3.6).

### 4.3.2 Flight Control Options

MATG supports various kinematic integration modes specified by *Flight Control* settings. Most important of these settings are described below.

The *wpCaptureMode* defines the aircraft behavior at waypoints. It has two options: *WP_FLY_BY* (default) and *WP_FLY_OVER*. Figure 3 illustrates the difference between *WP_FLY_OVER* and *WP_FLY_BY* capture modes. The *WP_FLY_BY* option, which is usually used, indicates the "turn inside" the waypoint, shown in Figure 3a). In this case the aircraft begins to turn before the waypoint position and completes the turn after its course is pointing at the next waypoint. Therefore, the trajectory doesn't cross any waypoints. In rare cases when it's desired to delay the turns until the aircraft has crossed the waypoints, the wpCaptureMode should be set to WP_FLY_OVER, as shown in Figure 3b).

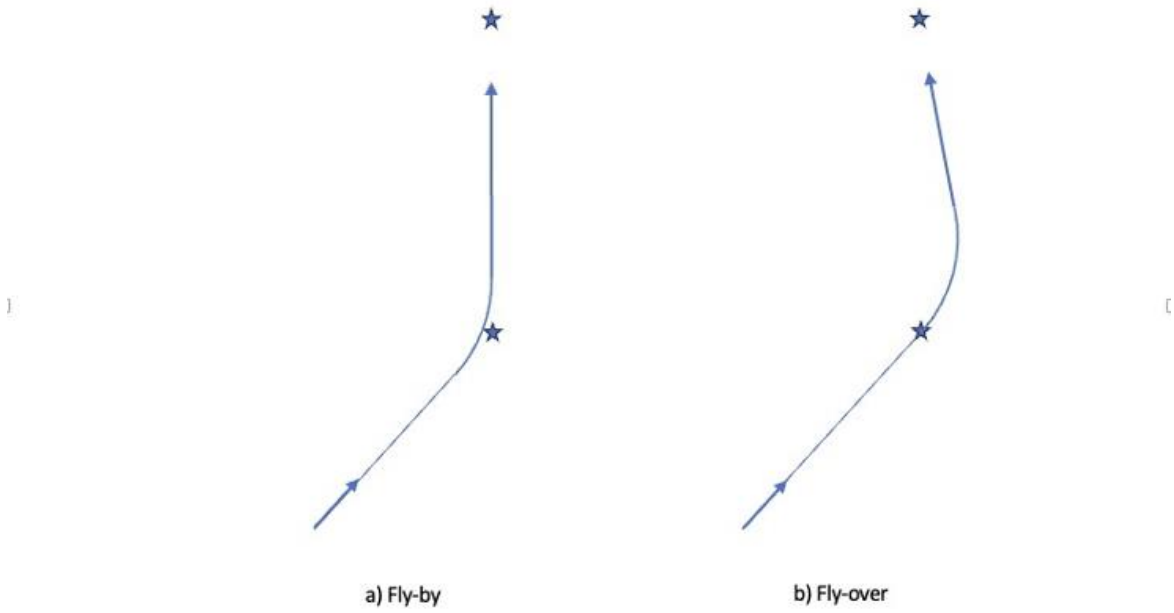



*Figure 3: Fly-by and Fly-over waypoint capture modes*

The Boolean *useAllWaypoints* setting determines how strictly MATG will follow upcoming waypoints in the constraints list. If it is set to false (default), MATG tries to automatically determine the next waypoint to be captured as the first constraint with position "in front of aircraft," meaning that the angle between the direction to this waypoint and the course of aircraft does not exceed 90 degrees (see Figure 4). All waypoints "behind

aircraft" will be "missed" (skipped) even if they are included in flight plan. The only exception is a constraint collocated with *destination* (if *destination* is set), which is always treated as if it is "in front of aircraft." This is needed to ensure that an aircraft will never miss its destination. This option can be useful if *constraints* may include both the past and future waypoints relative to initial aircraft state. In contrast, if *useAllWaypoints* is true, the MATG tries to capture all waypoints in the order listed in *constraints*. This option can be used if it is known that *constraints* represent the future waypoints only. This may require an external constraint management logic that would promptly remove the past waypoints from *constraints*.

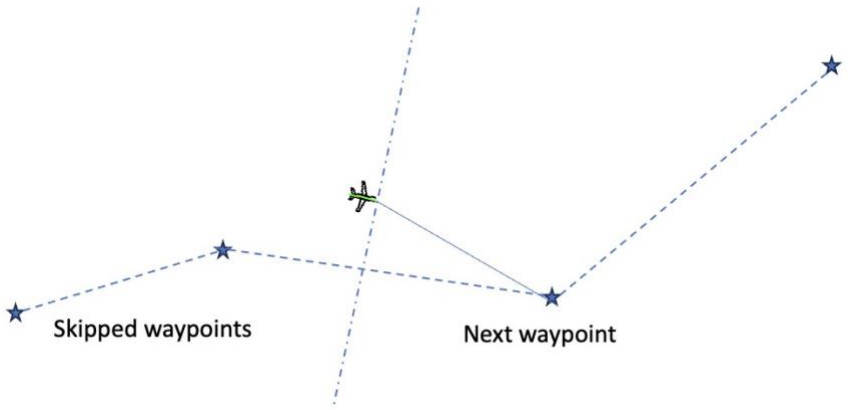



*Figure 4: Finding next waypoint for useAllWaypoints set to false*

The Boolean *useFixedFpaDescent* setting controls how MATG models descent segments. The default option for *useFixedFpaDescent* is false, meaning that aircraft will limit its descent rate to the value specified in APM, which usually corresponds to idle descent. In this case, the aircraft quickly descends to target altitude and maintains it until the next altitude constraint. Setting *useFixedFpaDescent* to true will ensure using a smooth descent profile with near-constant Flight Path Angle (FPA), as typically used for final descent to destination, for all descent segments.

The Boolean *useEnhancedSpeedControl* setting defines speed control logic in cruise. If it is set to true, the default option, an aircraft will not decelerate to target speed immediately in order to avoid unreasonably low speed. This is usually the desired behavior. If *useEnhancedSpeedControl* is set to false, an aircraft will always decelerate immediately to target speed and maintain it until it reaches another speed constraint. Note that *useEnhancedSpeedControl* affects only deceleration logic in cruise. Irrespective of this setting, an aircraft always accelerates immediately to target speed, and it always decelerates immediately to target speed in climb and descent segment (with the exception for Final Approach and Landing as described in Subsection 4.3.6).

The Boolean setting *stopAtLastConstraint* determines when trajectory generation should end. If it is set to false (default), the trajectory will end at specified time. If by this time all constraints are captured, the trajectory will extend beyond last constraint and continue with constant velocity vector from the last constraint. There could be situations when setting the time limit is the only sensible way to define the end of a trajectory, but this time limit is not always known or easy to determine. Therefore, it can be desirable to set *stopAtLastConstraint* to true. In this case a generated trajectory will be limited by the last captured constraint and stop there. This option is most useful for generating full trajectories from origin to destination.

### 4.3.3 Lateral Guidance

If *Next Constraint* has an *xy* position denoted as ($x^*$,$y^*$), MATG determines the target course as the following:

$$\theta^* = \arctan\left(\frac{x^* - x(t)}{y^* - y(t)}\right) \tag{4}$$

Then it changes the *true course* $\theta$ by turning it in direction of $\theta^*$ until the difference between them by modulus becomes lower than $\dot{\theta}\Delta t$, where $\Delta t$ is the integration time step set by a caller, and $\dot{\theta}$ is the rate of turn expressed by equation (5):

$$\dot{\theta} = \frac{g \tan \phi}{u} \tag{5}$$

where $g$ is the gravitational constant and $\phi$ is a bank angle known from APM.

Equation (4) is used for any *wpCaptureMode*. However, if *wpCaptureMode* is *WP_FLY BY*, lateral guidance includes additional turn detection logic. This logic starts with estimating the rate of turn from expected average *ground speed* during the turn. This rate of turn is used to estimate when the turn should start before the *Next Constraint* to ensure that after the turn the *ground course* is pointing at the next waypoint after the *Next Constraint*. The only difference from the *WP_FLY OVER* case is that after the beginning of the turn, MATG uses ($x^*$,$y^*$) from the next waypoint after the *Next Constraint* in Equation
(4).

Note that while executing the turn, MATG checks whether it can complete the turn before the waypoint after the *Next Constraint*. If this is not possible (e.g., aircraft is going too fast or waypoints are spaced too close to each other with sharp turns at some of them), MATG can skip the "unreachable" waypoint instead of capturing it. This has an important implication. MATG adds only the captured constraints to the list of *updated constraints*. Therefore, if there are skipped "unreachable" waypoints, some of *constraints* in *initial flight* will not be included in *updated constraints*.

If *Next Constraint* does not include *xy* but includes *true course* or *true heading*, the target course $\theta^*$ is set to *true course* or, if *true course* is also not known, it can be estimated from *true heading*. The logic of updating the *true course* from target course remains the same.

### 4.3.4 Speed Control Logic

If *Next Constraint* doesn't include any speeds (i.e. ground speed, TAS, CAS, and Mach), MATG assumes that it inherits speeds from a previous constraint. If only ground speed is known, MATG calculates a target TAS using vertical speed and wind data at the location of this target if available.

If CAS and Mach are not explicitly specified, MATG tries to hold constant TAS in cruise and use target CAS climb and descent (constant CAS capture). In this case a target CAS is calculated from target TAS and altitude.

If target CAS is explicitly specified, MATG assumes constant CAS capture and calculates a target TAS from target CAS and altitude. If target Mach is explicitly specified, MATG assumes constant Mach capture and calculates a target TAS from target Mach and altitude. Therefore, it can be assumed that a target TAS is always known for any speed capture conditions.

In most cases, if the difference between current state TAS $w(t)$ and target TAS $w^*$ by modulus exceeds the maximum possible speed change for one time step $|a\Delta t|$, the TAS for the next state is calculated as

$$w(t + \Delta t) = w(t) + a\Delta t \quad (6)$$

where $a$ is acceleration if $w^* > w(t)$ or deceleration otherwise defined in APM (note that deceleration is a negative value).

If $|w^* - w(t)| < |a|\Delta t$, MATG sets $w(t + \Delta t) = w^*$.

If target CAS and/or Mach were explicitly specified, the CAS and Mach for next state are recalculated from updated $w(t + \Delta t)$.

In climb and descent, aircraft behavior in either acceleration or deceleration is the same as described above. This means that the aircraft starts accelerating or decelerating to target speed immediately.

In cruise, if *useEnhancedSpeedControl* is set to true, an aircraft will start decelerating to target speed only when an estimated distance to target $d(t)$ is below an estimated minimum distance for deceleration:

$$d_m = \Delta T_m * (u(t) - a * \Delta T_m/2) \quad (7)$$

where $\Delta T_m$ is an estimated deceleration time to a target speed $u^*$:

$$\Delta T_m = (u(t) - u^*)/a \quad (8)$$

This is a default behavior that models what pilots would most likely do, starting deceleration only when necessary rather than reducing speed prematurely long before needed.

In practice, $d(t)$ is calculated along a straight line, so a slightly higher value of $d_m$ is used to account for a possible curvature of flight path between the current state and target positions.

The same approach is used for deceleration to landing speed as described in subsection 4.3.6.

### 4.3.5 Altitude Control Logic

The first step in altitude control logic is to determine the current flight phase. If *Next Constraint* does not include *altitude*, MATG assumes the cruise flight phase; otherwise if target altitude exceeds the current altitude (plus small tolerance), this is considered a climb phase; if the current altitude exceeds target altitude (plus small tolerance), this is considered a descent phase.

For a climb phase, MATG uses the rate of climb from APM as a maximum possible value for vertical speed. For a descent phase, MATG uses the rate of descent from APM as a minimum (maximum by modulus) possible value for vertical speed.

These rates of climb and descent can be specified in APM as constant values or as altitude-dependent tables, similar to BADA PTD-tables [2]. In the latter case, MATG determines a maximum/minimum value of vertical speed by interpolating from these tables to the current altitude. This maximum/minimum value becomes a target vertical speed $v^*$ for a climb or descent segment, which plays the same role as $u^*$ for speed control described in subsection 4.3.4.

MATG uses vertical acceleration and deceleration determined from load factors specified in APM for smooth transitions between zero and $v^*$.

By default, an aircraft starts climbing and descending to target altitude immediately, which is similar to speed control logic in climb and descent segments described in subsection 4.3.4. This implies that if the rate of descent in APM corresponds to idle descent, this is what MATG will use for all descent segments. The consequence of this is that for long descent segments an aircraft will reach the lower target altitude and maintain it until the next altitude constraint.

This behavior is typical for traditional airliners, but it is not always desirable. One exception is the final approach when aircraft has to carefully adjust its FPA depending on its speed in order to land at the destination and not earlier (see subsection 4.3.6).

The same near-constant FPA descent logic could be used for continuous descents in other descent segments by setting *useFixedFpaDescent* to true. In this case, a vertical speed is calculated as

$$v(t+\Delta t) = max\left\{u(t+\Delta t)\frac{z^* - z(t+\Delta t)}{d(t+\Delta t)}, v^*\right\} \tag{9}$$

where $z^*$ is a target altitude.

Note that for descent segments $z^* < z(t + \Delta t)$, and both $v(t + \Delta t)$ and $v^*$ are always negative.

### 4.3.6 Final Approach and Landing Procedures

If *Next Constraint* is a *destination*, it needs special treatment for the following reasons:

1. the aircraft has to land precisely at a specified position *xy* and *altitude*.
2. the aircraft has to decelerate to its landing *ground speed* at the time of landing.
3. if landing is not physically possible because the altitude and/or speed is too high, MATG has to handle this situation as a missed approach and make another attempt to land.
4. if landing speed is near zero, MATG must model a vertical landing procedure.

To address the first issue, MATG uses fixed FPA descent in final approach by constantly adjusting its vertical speed as described by Equation (9) in subsection 4.3.5 irrespective of *useFixedFpaDescent* setting.

To ensure that the aircraft decelerates to a desired landing speed, the variant of enhanced speed control is used in final approach as described in subsection 4.3.4 irrespective of *useEnhancedSpeedControl* setting.

Handling a missed approach depends on what is causing the situation. One possibility could be that the aircraft has enough time to decelerate to its landing speed before the destination, but its altitude at this time would remain too high even if the highest possible, according to APM, rate of descent is used. If the landing speed is near zero, suggesting that the aircraft is capable of vertical landing, and estimated AGL altitude at *destination* doesn't exceed a threshold for safe vertical landing $\Delta z^*$, MATG uses the vertical landing procedure. If, however, a predicted AGL altitude at *destination* is higher than this threshold, MATG prevents deceleration to zero speed and treats this situation as a missed approach.[1]

It is also possible that the aircraft can't land because it doesn't have enough time to decelerate to landing speed. In this case MATG will set a target vertical speed $v^*$ to zero, effectively blocking descent until the aircraft is able to decelerate to landing speed. This would also lead to a missed approach and another attempt to land.

Note that for a realistic flight plan, a missed approach is unlikely to happen. However, MATG tries to do anything possible to prevent failures, and this requires providing low level control logic for generating a safe and reasonable trajectory under all possible conditions.

[1] Currently, $\Delta z^*$ is hard-coded to 100 ft, and missed approach is handled simply by setting *destination* as *Next Constraint*, so the aircraft turns back to it to make another attempt to land. This logic can be adjusted when missed approach procedures for eVTOL are published.

Finally, MATG models transition to a vertical landing procedure by setting the rate of descent to a low value defined in APM when AGL altitude falls below $\Delta z^*$, and gradual deceleration to zero speed.

### 4.3.7 Execution of Flight Commands

Some flight commands described in subsection 4.2 cannot be executed in preprocessing because they have *Conditions* or *Actions* referring to specific aircraft states. An example of such command is:

**Use cruise speed between waypoints**

*Conditions* State Distance to Target > 1 nmi

*Actions* USE State TAS = 120 knots

The intent of this command is to use a cruise speed of 120 knots between waypoints (if it was below this value) if the aircraft distance to next target exceeds one nmi.

To execute such commands, MATG checks at every time step if there are any active commands with valid *Conditions*. If such commands are found, MATG redefines *Next Constraint* from this command's *Actions*.

For instance, for the command "Use cruise speed between waypoints", MATG will set TAS for *Next Constraint* to 120 knots. This will effectively cause an aircraft to accelerate to 120 knots and maintain this speed as long as $d(t)$ remains higher than one nmi. Note that although the command says "use State TAS = 120 knots", it will not instantly change the state TAS from the lower value to 120 knots, since this would violate the flight physics. MATG will still use its normal speed control logic described in subsection 4.3.4.

Once this condition is no longer valid, MATG will use *Next Constraint* from initial *constraints* with initial values for all variables including TAS. If this initial TAS is lower than the cruise speed of 120 knots, an aircraft will decelerate to this target TAS as it would normally do without flight commands.

It is possible that more than one command is active at the same time. For instance, one command may require using a specific speed, and another command may require using a specific target altitude. In this case, both commands are applied to *Next Constraint*, so its speed and altitude will be set from active commands.

# 5 MATG Implementation and Performance

MATG in a narrow sense is a class implementing a *TrajectoryPredictor* interface. This interface and implementation rely on data structures and services provided by the **common** package in JADEM (bold font is used to indicate a package name). Some of them are potentially useful for other applications not related to trajectory predictions. Examples include variables with units providing easy conversions between different units of measurement, logging/debugging/tracing facility, validation code checking for possible inconsistencies and errors in data, grid mapping for fast search of closest neighbors in two-dimensional space, and of course, all data structures representing *Flight Data* described in Subsection 4.1. The **common** package also includes code for loading and preprocessing *Flight Data* from various data sources, and for writing *Flight Data* in formats readable by humans and other applications.

To allow using MATG outside JADEM, a separate library was created, including a *TrajectoryPredictor* interface, its MATG implementation, and the whole **common** package from JADEM. This relatively small library can be called like any other Java library by any application that needs to generate and/or analyze trajectories. For instance, the ETAG service calls MATG to estimate the times of arrival.

The MATG library can also be called as a standalone application. This is useful for testing MATG and flight input data. This MATG application, MatgMain, can be viewed as a lightweight variant of SaaControl described in Section 3. It uses similar configuration files, it can load the same flight data files, set user-defined control

options, execute flight commands, and do other preprocessing if needed. It also generates trajectory files viewable in DTV. It detects and reports errors and generates logging/debugging output files.

MatgMain also reports timing information about MATG performance. Naturally, the physical time of trajectory generation depends on hardware, time step, and the length of trajectory. For typical UAM flights with a one second time step, MATG generates one trajectory in half of a millisecond. This means that it can easily generate trajectories for several thousands of flights in just a few seconds. This makes MATG suitable for modeling high density traffic and conflict management that typically requires generating multiple trial trajectories.

# 6 Flight Modeling and Data Analysis Methods

## 6.1 Sparse Representation of Noisy Trajectory Data

Certain air traffic modeling tools and research projects need to use Instrument Flight Rules (IFR) and Visual Flight Rules (VFR) flight data from historical ground-based radar tracks [17]. There are several problems with using this data. The tracks can be noisy and recorded at arbitrary times. The size of track data files can be large resulting in high storage and memory requirements for processing this data. Some simulation tools, such as Fe$^3$ [24] and JADEM [4], require flight plans (intent) and APM as a part of flight input data, which may not be available from recorded tracks.

To address these issues, a software tool for sparse representation of noisy IFR/VFR trajectories, named C3FlightModeler, was implemented at NASA Ames. It allows the extraction of intent and some aircraft performance parameters from recorded track data. After that, the noise-free IFR or VFR trajectories can be generated from flight plans and APM in the same way as trajectories for simulated UAM or other flights.

The tool is based on the Cumulative Change Cost Function (C3F) algorithm. The normalized Cumulative Change Cost Function is defined as

$$C3F = C_{Speed} + C_{Course} + C_{FlightPhase} + C_{Time} \tag{10}$$

Each term in equation (10) is defined in the same way:

$$C_{Value} = ValueChange/ValueChangeLimit \tag{11}$$

where $C_{Value}$ is a Change Cost for any *Value*, such as *Speed*, *Course*, *FlightPhase*, or *Time*, and *ValueChange* is a change in *Value* between last trajectory state *LastTR* and last Trajectory Change Point *LastTCP*.

$$ValueChange = |Value(LastTR) - Value(LastTCP)| \tag{12}$$

Once $C3F$ exceeds 1, the *LastTR* is considered as new Trajectory Change Point (TCP), which is added to *FlightData.constraints*.

The algorithm effectively removes high-frequency noise from input trajectory data. It requires no more than two subsequent trajectory states, reducing memory use and eliminating the need to store the entire loaded trajectories.

The *ValueChangeLimit* in equation (11) is a scale factor that defines a sensitivity of C3F to this *Value*. These scale factors are user-defined, with the following default values: *speedChangeLimit* = 30 knots, *courseChangeLimit* = 15 deg, and *timeChangeLimit* = 300 sec.

*FlightPhase* is defined numerically as -1 for *verticalSpeed* < −*verticalSpeedLimit* (descent), +1 for *verticalSpeed* > *verticalSpeedLimit* (climb), and 0 for cruise. Therefore, it is assumed that *FlightPhaseChangeLimit* = 1, resulting in new TCP for every change in *FlightPhase*, such as Top of Climb or Top of Descent. The *verticalSpeedLimit* is another user-defined threshold that defines a sensitivity of C3F algorithm

to changes in vertical speed, with a default value 300 fpm. *verticalSpeed* can be estimated in two ways: from track data for vertical speed, if available, and from altitude changes between trajectory states. The algorithm determines which of these estimates results in less noisy *verticalSpeed* and uses it to determine the *FlightPhase*. Optionally the algorithm can set vertical speed to zero if the altitude difference between two consecutive trajectory states is below user-defined *altitudeChangeLimit*. This can be useful to remove altitude noise from recorded track data caused by transponders transmitting pressure altitude information in 100-foot increments.

The algorithm also includes the logic to determine TCPs for start and end points of long gradual turns. It is based on the condition of "being in turn" defined as *turnRate* > *turnRateLimit*, where *turnRate* is determined as the minimum between the rates of turn evaluated from *CourseChange* (Equation (12)) and from a course difference between two subsequent trajectory states. When an aircraft is in a turn, the *courseChangeLimit* in Equation (11) is replaced with 180 deg so that detection of next TCP is far less likely until the condition *turnRate* <= *turnRateLimit*, indicating that the aircraft is no longer turning, is met. The default value of *turnRateLimit* is 3 deg/sec, but this value may need to be increased for flights with long or multiple consecutive turns, as in loitering patterns, when defining the turn begin and end points becomes problematic.

Before applying the main logic of the C3F algorithm expressed by equations (10-12) with corrections described above, C3FlightModeler tries to remove the outliers from input trajectory states. This is accomplished by excluding states with horizontal or vertical speeds that would result in unreasonably high accelerations exceeding *accelerationLimit*, set to 30 fps/sec by default.

In addition to extracting flight intent as a sequence of TCPs (*constraints*), the C3F algorithm tries to estimate aircraft performance parameters from recorded track data. It uses typical default values for horizontal accelerations and decelerations, rates of climb and descent, and bank angle as a starting point. If an aircraft accelerates or decelerates during the flight, the APM parameters for acceleration and deceleration are calculated as maximum values of accelerations and decelerations inferred from speed changes between trajectory states. The rates of climb and descent are estimated similarly from the less noisy vertical speed as described above.

The bank angles for TCPs representing the start points of sharp turns are estimated as maximum bank angles based on turn rates determined from course differences between two subsequent trajectory states (Figure 5). These course differences correspond to the angles between two consecutive segments in the original trajectory, shown as blue lines (with grey lines indicating the directions of these segments beyond their end points). The resultant bank angle is further adjusted in two ways. If a segment in the original trajectory is a part of an arc (light blue curve), the bank angle calculated from the turn rate would correctly reflect the course along this segment, but it would result in a trajectory increasingly deviating from the arc (red dashed arrows). So the turn rate has to be doubled to ensure that aircraft can cross the end point of this segment and rejoin the arc as indicated by a red solid arrow. This is roughly equivalent to doubling the bank angle, and this is what the algorithm does for simplicity. Having a slightly higher bank angle is better than underestimation of bank angle that would not allow following the curved trajectory. A second adjustment is needed to prevent unrealistically high values of bank angle.

Finally, the algorithm uses the default bank angle of 15 deg as a minimum possible value.

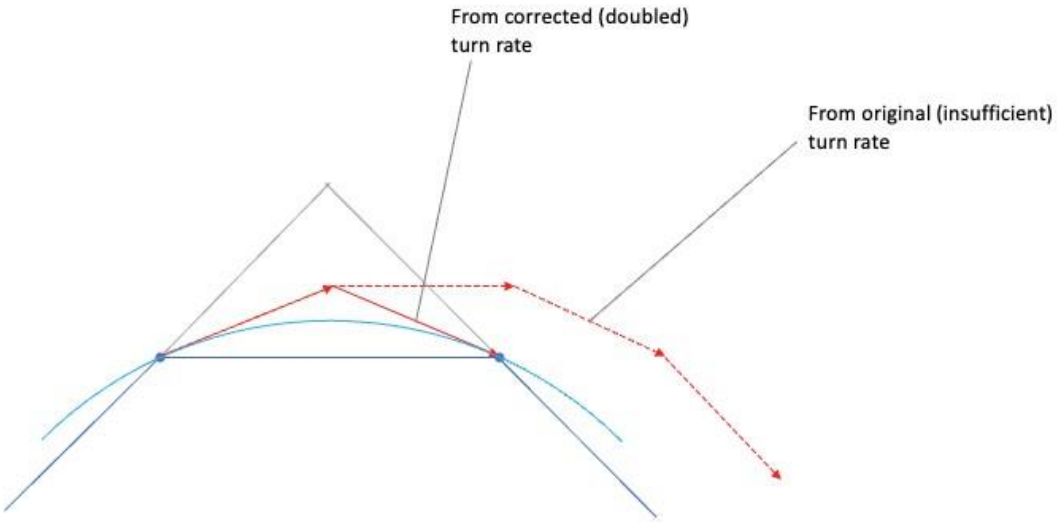



*Figure 5: Turn rate estimation in C3F algorithm*

## 6.2 Automated Trajectory Comparison

The Flight Comparison tool (FlightComparer) was developed as a separate project to simplify and partially automate comparisons between trajectories. It was based upon existing manual methods of converting differently formatted flight data files into a common format, and then combining the results together for comparison. This required an approach that solves two problems: a common way to read and represent flight data coming from different sources and in different formats, and a tool that could easily take that data and perform a variety of analyses on it.

The first problem was solved by defining a standard *Flight Data File* object which could be used as a basis for reading, writing, and otherwise handling flight data files in terms of both the flight data they contain and metadata regarding each type of file. Using this standard, handlers, as child objects, can then be created to implement all necessary functions relevant to each kind of flight data file of interest. These objects can be stored in a library to be called as needed, and new handlers can be created and added as need arises to support new flight data file types.

The second problem was addressed by separating the analysis tool into two parts: a computational library, and an interface. The computational library is a collection of computational and supporting functions that perform needed analysis. Primary functions will accept flight data and any other needed parameters, and return the results, generally in list format, to be output. Supporting functions may be used by either the primary functions or the interface for performing additional calculations or for organizing or formatting the data. The interface interprets and validates user input, reads in specified file(s), calls requested analysis, and formats and outputs the results. Currently this part is implemented as a command line interface, but in the future it could be reengineered as a graphical user interface (GUI). By separating the functionality this way, new analysis methods can be added to the library, and then with limited effort, supported in the interface (new flight data file types will also need to be added in a similar way to be supported by the interface).

The Flight Comparison tool currently supports four types of analysis: simple route duration computation, ETA calculation to route waypoints, horizontal distance analysis between two flights (flight separation), and along-track route comparison between two flights.

Route duration is a simple computation of the total flight time of a given route based on the first and last points of its trajectory. This is useful as a base level comparison of the performance of a trajectory computation and can be used to compare with the duration of similar trajectories computed by other trajectory generators.

ETA calculation to route waypoints can be done with flight data that contains at least some named trajectory points. These named points represent either actual or closest approximation of constraint waypoints used to create the flight trajectory. Then the ETA, or flight time, to any waypoint is the difference of the flight time to that waypoint and the flight time to the first waypoint. The output includes the list of waypoints for each route along with the list of flight times to each waypoint. Multiple flights on the same route are also listed alongside for comparison. This gives a more detailed analysis of the progress of a flight along its route as well as the ability to compare it with other flights on the same route.

Horizontal distance analysis requires specifying two flights, a reference flight and a comparison flight, though these designations are generally arbitrary for this analysis. These flights are assumed to be on a similar route, though the function neither requires nor enforces this. What is required is that the time range of both trajectories must have some overlap. For each time step that both trajectories share, the relative distance between both flights is computed. Figure 6 demonstrates one such computation at time $t_i$. This results in a list of distances that can then be analyzed statistically to produce output that includes maximum distance, average distance, and standard deviation of distances. The results of the horizontal distance computations will be affected by the start time of each trajectory, so this should be considered when performing this analysis. In most cases it will probably be desirable to have both trajectories start at the same time, so the tool includes an option to automatically sync the start times of both flights. This type of analysis is useful for providing a general indication of how closely two flight trajectories match each other throughout the route, though it does not indicate when one flight is ahead of or behind the other.

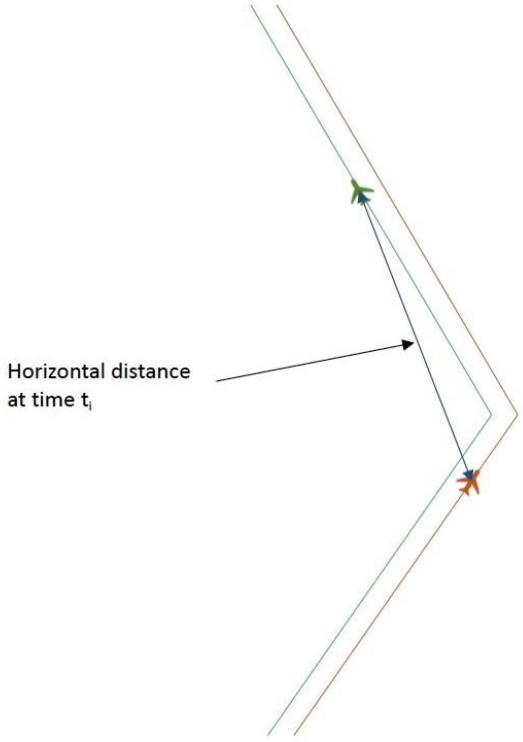



*Figure 6: Horizontal distance between two flights*

Along-track analysis also requires specifying a reference and a comparison flight. In this case the reference flight is used to drive the analysis, and the results will be based on the comparison flight. As is the case with horizontal distance analysis, it is assumed that both flights are on a similar route, though this is not required, and there must be some overlap in the time range. This computation simplifies both trajectories by computing the distance between successive points of each trajectory and converting them into one dimensional arrays of distance traveled over time. For trajectories with a small enough time step, such as one sec, this should be a reasonable approximation.

To compare the relative along-track distance traveled for each flight, the fractional progress is used. For each time step $t_i$, the fractional progress of a flight, $P_i$, is computed as follows:

$$P_i = d(t_i)/D \tag{13}$$

where $d(t_i)$ is the distance traveled at time $t_i$, and D is the total distance traveled. This fractional progress is first computed for the reference flight to determine how far along its route the reference flight has traveled at time $t_i$. Then the reference flight's fractional progress is applied to the comparison flight to determine the equivalent distance traveled for the same fractional progress along its route. The result of this computation is the equivalent distance $d(t^*)$ where time $t^*$ is the time at which the comparison flight would be at the same fractional progress as the reference flight is currently. The value for $t^*$ is simply interpolated from the distance traveled over time array of the comparison flight.

Figure 7 shows the positions of reference and comparison flights at time $t_i$. The fractional progress, $P_i$, is computed for the reference flight and, when applied to the comparison flight, would put it at distance $d(t^*)$ and time $t^*$ as shown. This allows a difference in distance to be computed from $d(t_i)$ and $d(t^*)$, indicated by the blue segment in the figure, as well as a difference in time from $t_i$ and $t^*$. A negative difference indicates that the comparison flight is behind the reference flight at time $t_i$ while a positive difference indicates that the comparison flight is ahead of the reference flight. The result is two lists of differences, one of distances and the other of times, that can both be analyzed statistically.

Similar to the horizontal distance computation, the results can be affected by the choice of start time for each flight, and the start times may be automatically synced if desired. For two flights on similar routes, this analysis has the advantage of providing a measure of how far the comparison flight was ahead of or behind the reference flight on its route relatively speaking, although it does not, itself, indicate the similarity of the two routes.

The Flight Comparison tool allows analysis of single flights or flight pairs, but it also supports group or batch data entry to allow multiple flights to be processed at once using up to two different types of flight data sources. For some types of analysis, additional group or aggregate data may be provided.

The ETA calculation to route waypoints, as described previously, can process multiple flights on multiple routes. Different routes are output separately, but flights on the same route are displayed together in multiple columns. Additional columns are also added to show the difference in ETA times between successive flights and the first listed flight.

Along-track and horizontal distance analyses support a "batch mode" of data input. Groups of flights can be linked together with flight IDs that match specified formats and compared as reference and comparison flights. For example, if a set of flight data contained flights with IDs SRC1-ABC100 - SRC1-ABC199 and another set of flight data contained flights with IDs SRC2-ABC100 - SRC2-ABC199, then flights starting with "SRC1-" could be specified as reference flights and flights starting with "SRC2-" could be specified as comparison flights. Flights with matching base IDs such as SRC1-ABC100 and SRC2-ABC100 would be paired and analyzed. In

addition to the individual statistical results from each flight pair, aggregate statistics, consistent with the type of analysis, are also provided including values such as average maximum flight separation, average mean time difference, and average standard deviation value.

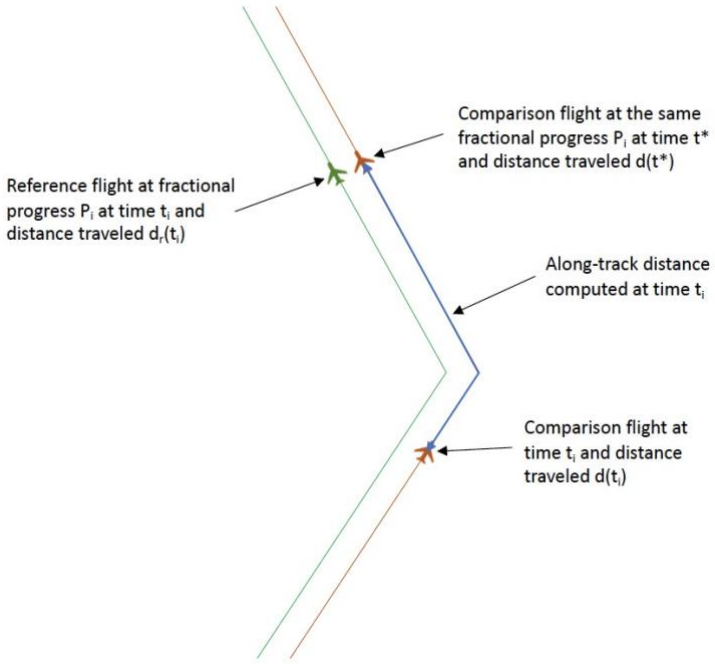



*Figure 7: Along-track distance between reference and comparison flights*

# 7 MATG Verification and Validation

## 7.1 Applications to UAM Flight Modeling

One common use of MATG is to generate trajectories of simulated UAM flights in JADEM [17]. An example used in this paper is for UAM traffic scenarios in DFW TRACON Airspace based on the airspace route structure and traffic demand developed by Virginia Tech Air Transportation Systems Laboratory using a mode-choice model for commuter trips, given a set of socioeconomic factors and historical commuting patterns [21]. The input traffic scenario included a set of forecast trips between origin and destination vertiports and a set of flight plans that are defined by a sequence of waypoints (latitude, longitude, and altitude). Each trip included planned departure time, and planned cruise altitude and speed. For some waypoints in flight plans the altitudes were not provided. All flights were assumed to be VTOL, but speed profiles for entire flights were not specified. Therefore, JADEM had to include the UAM flight modeling logic that could fill in the gaps in flight plans, using planned cruise altitudes and speeds and certain heuristics to ensure that the flight plans for MATG are "flyable".

The MATG-generated trajectories for a subset of UAM flights were compared with trajectories generated by Fe$^3$ for the same scenario. These comparisons are valuable because MATG used a simplified kinematic model of a six-passenger quadrotor eVTOL [14], while Fe$^3$ could use either a BADA-based "kinetic" model or a higher

fidelity 6DOF quadrotor model. Some differences between MATG and Fe³ trajectories could be attributed to assumptions used by JADEM and Fe³ in their UAM flight modeling logic.

The following figures produced using DTV show comparisons for a few UAM flights in top view plots (on the left) and altitudes and speeds as a function of time (on the right). All plots include trajectories generated by MATG (red) and Fe³ (green with "FE3" prefix before the flight id), along with constraints from JADEM's UAM flight model used by MATG to generate these trajectories (blue with "CS" prefix before the flight ID). The original altitudes for all waypoints, including the undefined altitudes denoted as -999, are shown as inserts in top view plots.

For flight UFA1 the trajectories for both MATG and Fe³ with the Kinetic Model (Figure 8) appear very close to the waypoints and to each other on the top views. One exception is near the destination, where the Z259 waypoint is skipped by Fe³. Fe³ also deviates from altitude constraints and reduces speed twice during the cruise portion of the flight, resulting in a later arrival compared to MATG. All these differences could be explained by certain features of UAM flight modeling or control logic in Fe³. Another notable difference is a much longer vertical descent for Fe³ with Kinetic model. This is likely because Fe³ is using a lower Rate of Descent for landing close to 17 fpm, while 100 fpm is used in MATG according to recommendations from [20].

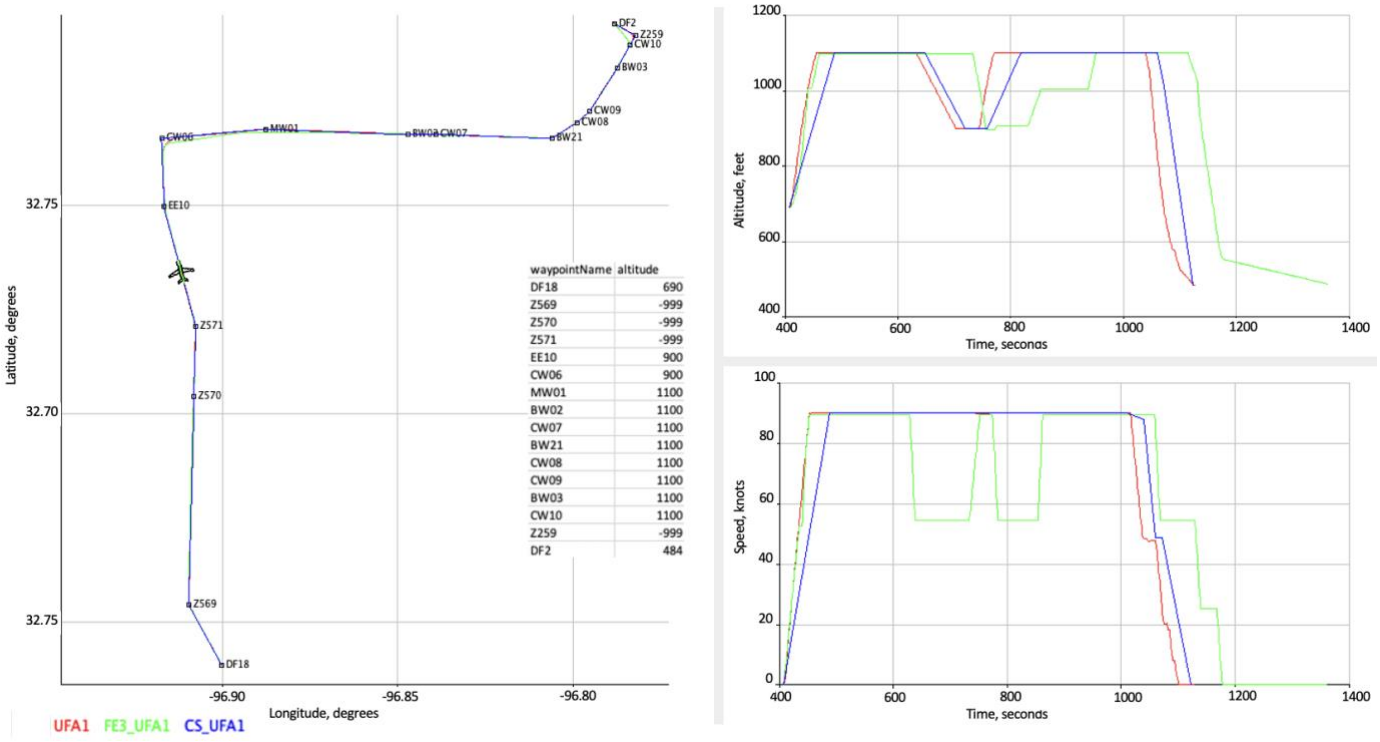



*Figure 8: UFA1 flight: MATG vs. Fe³ with Kinetic Model*

For the UFA65 flight the MATG trajectory on the top view follows the flight plan closer than for Fe³ with Kinetic Model (Figure 9), especially at sharp turns. Another notable difference is that Fe³ seems to climb closer to target altitude 1600 ft before descent. Figure 10, that includes a snapshot of aircraft positions according to MATG and Fe³, can help understand this difference.

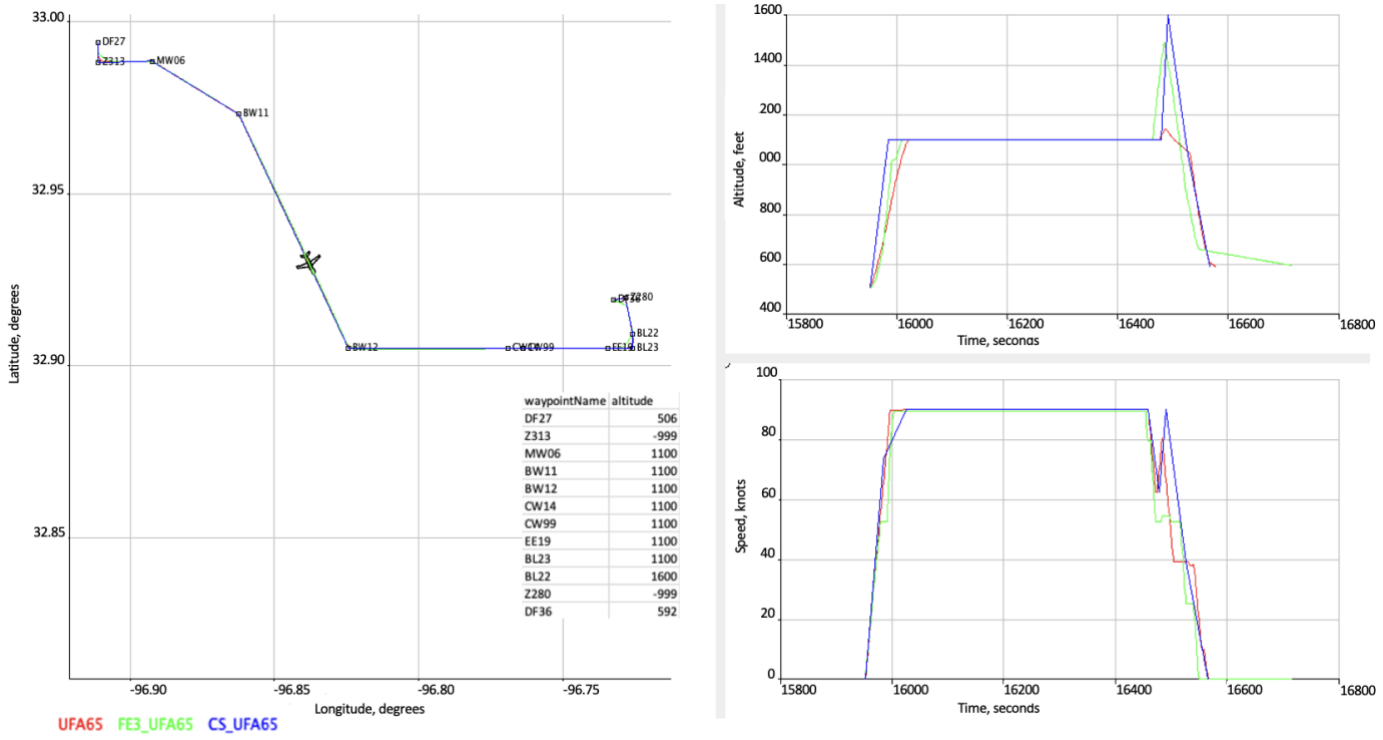



*Figure 9: UFA65 flight: MATG vs. Fe³ with Kinetic Model*

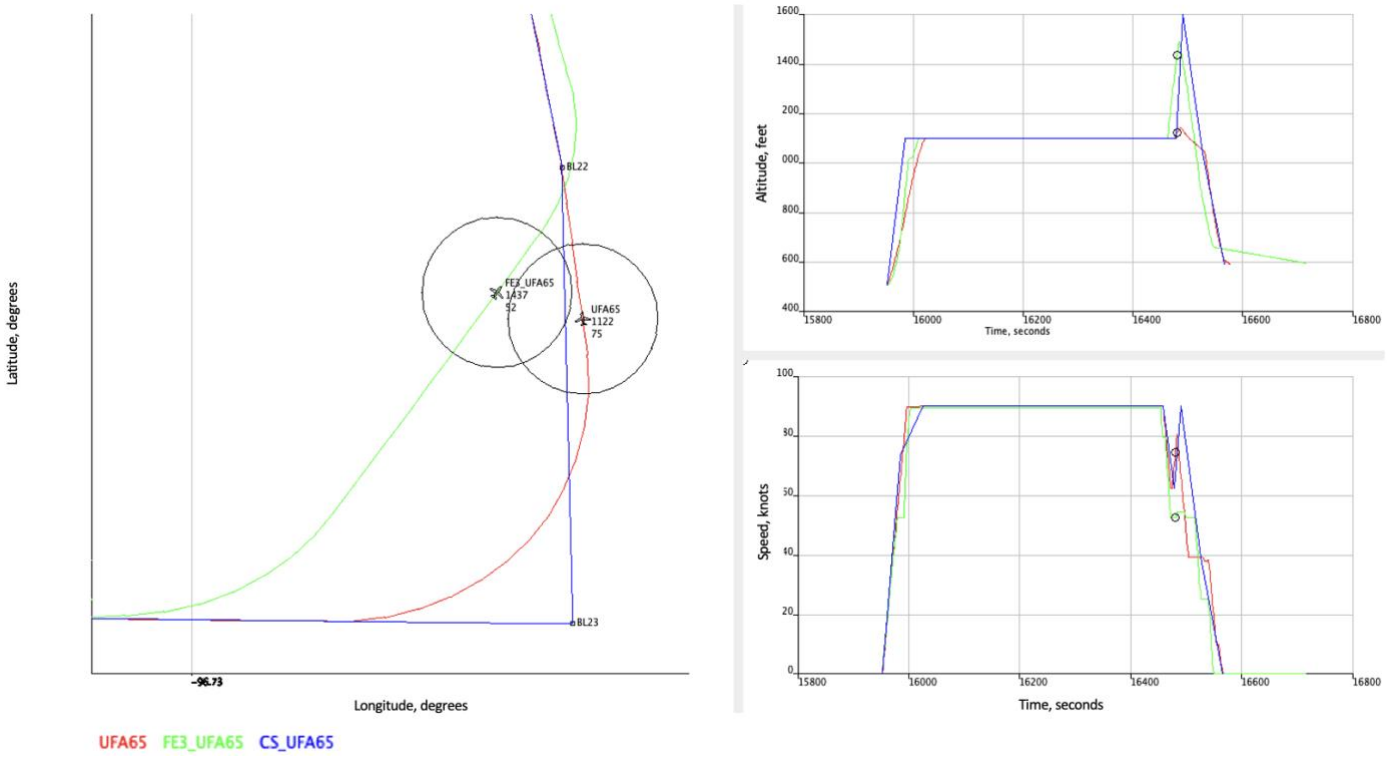



*Figure 10: UFA65 flight near BL23 waypoint: MATG vs. Fe³ with Kinetic Model*

As it was the case for UFA1 flight, Fe$^3$ skips the BL23 waypoint with altitude 1100 ft and goes straight to the next BL22 waypoint with higher altitude 1600 ft. MATG, in contrast, captures BL23 first and, only after the turn at BL23, starts climbing to 1600 ft. Therefore, Fe$^3$ allows more time for the aircraft to climb to 1600 ft as reflected in the altitude plot. Other than this, altitude and speed profiles look similar between MATG and Fe$^3$. The difference in flight duration would also be small if not for the lower final Rate of Descent used by Fe$^3$.

Figure 11 shows another case that illustrates the difference in turn behavior between MATG and Fe$^3$ with Kinetic Model, which in this case is most pronounced at the beginning part of the flight. Figure 12, including a snapshot of aircraft positions according to MATG and Fe$^3$, clearly shows that MATG captured all waypoints in order: Z260, BL20, BL19, and EE22. Fe$^3$ skipped the last three of these four waypoints while MATG was able to capture all of them because of a steeply reduced target speed, set to the value below 50 knots at BL19. This was a result of JADEM's UAM flight modeling logic that tried to reduce speeds at sharp turns. Skipping these three waypoints by Fe$^3$ could also explain the segment with constant altitude 1200 ft that does not correspond to any altitudes in flight plan and is missing in the MATG trajectory. Another difference between MATG and Fe$^3$ was an unexpected, short climb segment before the final descent. This could be because Fe$^3$ in its UAM flight modeling logic tried to use a planned cruise altitude, defined as 1600 ft for this flight, in the final portion of the flight where it skipped three waypoints (MW02, BW07, and Z356) and went straight to the destination (Figure 13). MATG didn't need to do this since the waypoints MW02 and BW07 had a well-defined altitude of 1100 ft.

Figures 14 through 16 show comparisons between MATG and Fe$^3$ with the 6DOF Model. It appears that the 6DOF model suggests a higher rate of initial climb, but the final descent portion looks more similar to MATG compared with the Kinetic Model. All differences in altitude and speed profiles due to the differences in UAM flight modeling logic remain for both Kinetic and 6DOF models. In addition, larger lateral deviations from the flight plan can be noted in the case of the 6DOF model, especially near the end of the flight and around sharp turns. The 6DOF model also results in speed oscillations that didn't occur in the case of the Kinetic model. These behaviors could be attributed to higher fidelity control logic for 6DOF model. It is not clear, however, if the same behaviors can be assumed for the six-passenger quadrotor eVTOL aircraft in Ref. 14.

As can be seen from these comparisons, UAM trajectories generated by MATG and Fe$^3$ are close to each other, especially when the 6DOF model is used. The only exception is a slightly larger average mean flight separation for the 6DOF model that can be attributed to larger lateral deviations possibly caused by higher fidelity control logic in Fe$^3$. In cases when MATG trajectories look more reasonable, this can be attributed to JADEM's UAM flight modeling logic rather than to the MATG kinematic model, which in all cases closely follows the flight plans. These comparisons provided sufficient justification for using JADEM with MATG for research that involved modeling UAM traffic [17].

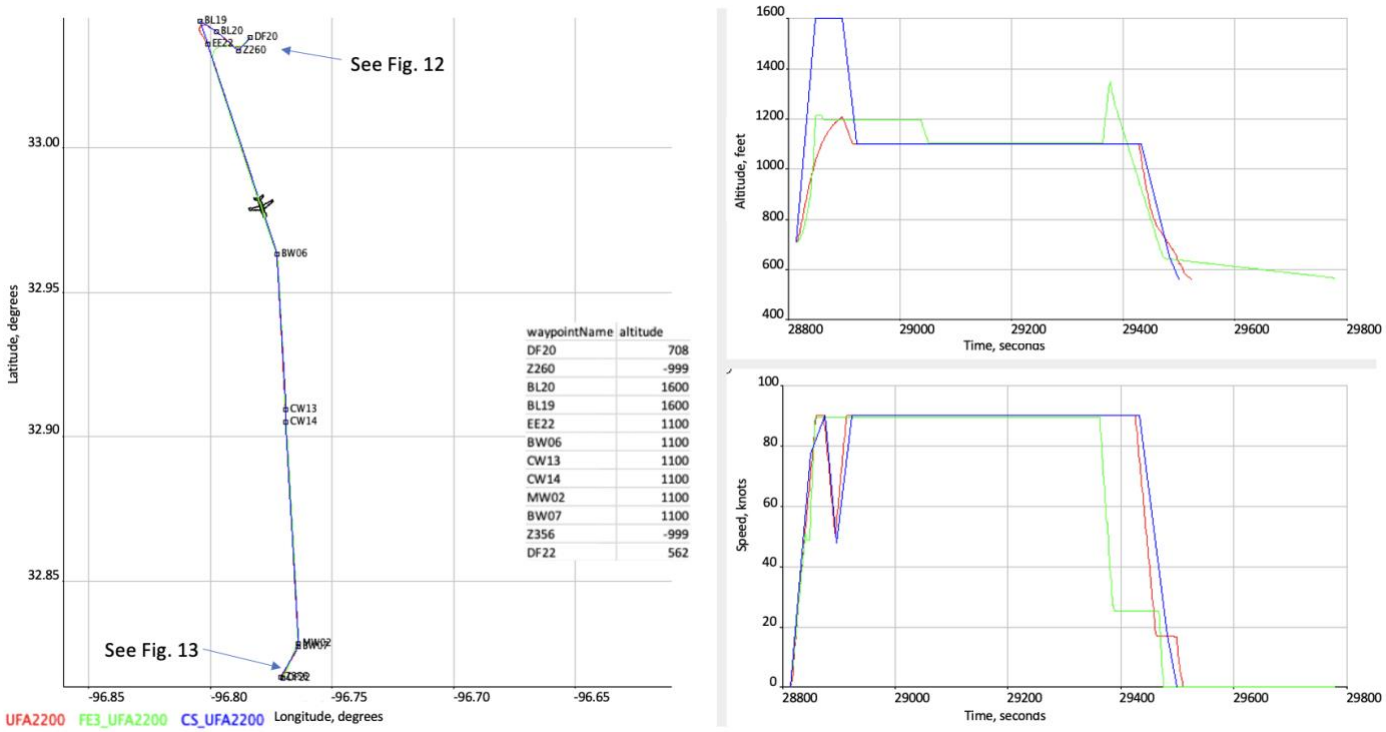



*Figure 11: UFA2200 flight: MATG vs. Fe³ with Kinetic Model*

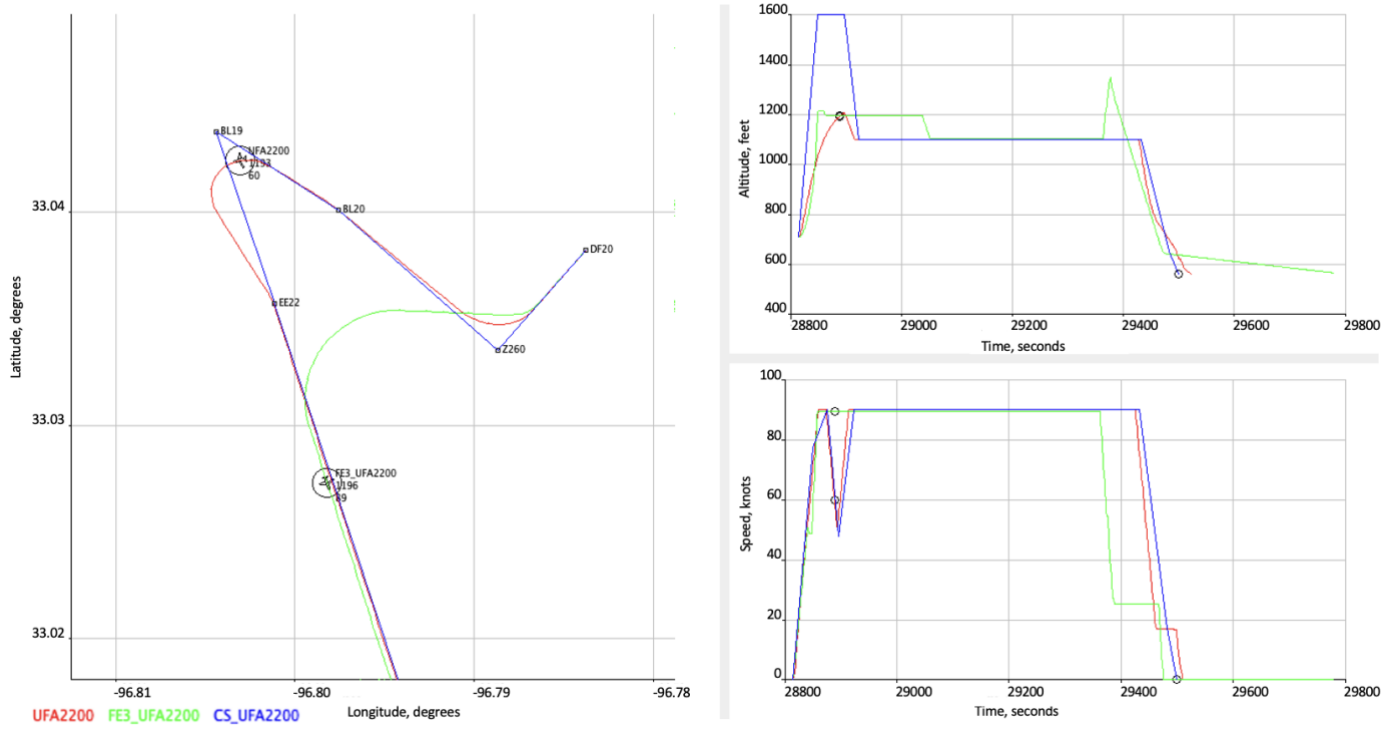



*Figure 12: UFA2200 flight near BL19 waypoint: MATG vs. Fe³ with Kinetic Model*

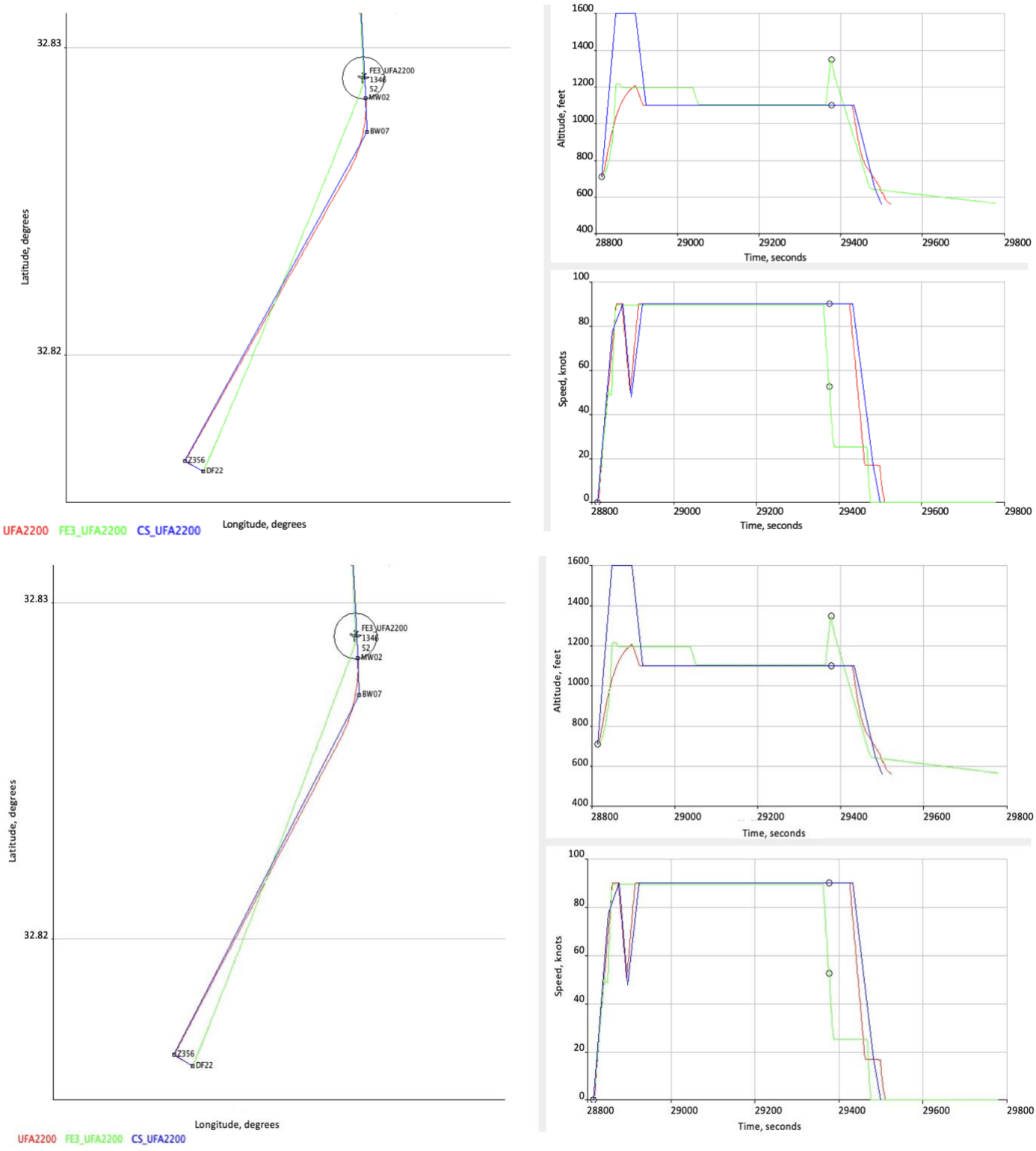



*Figure 13: UFA2200 flight near BW07 waypoint: MATG vs. Fe³ with Kinetic Model*

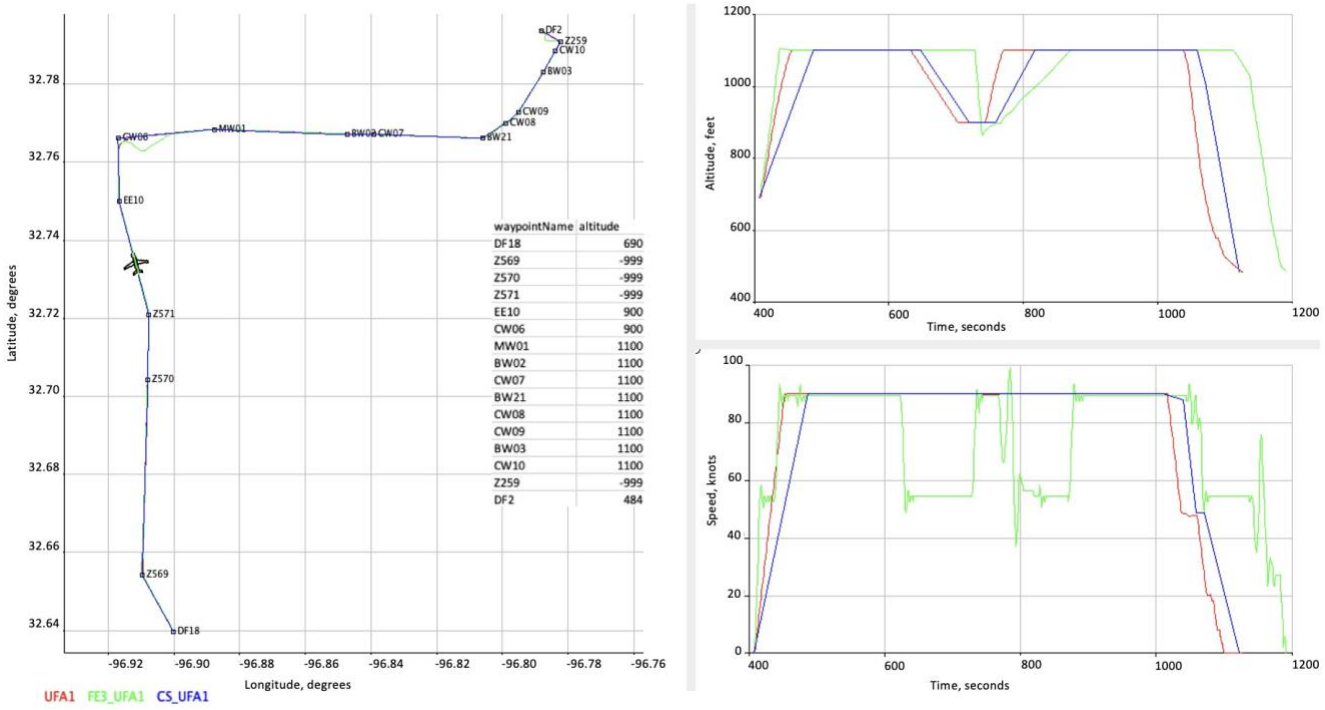



*Figure 14: UFA1 flight: MATG vs. Fe³ with 6DOF Model*

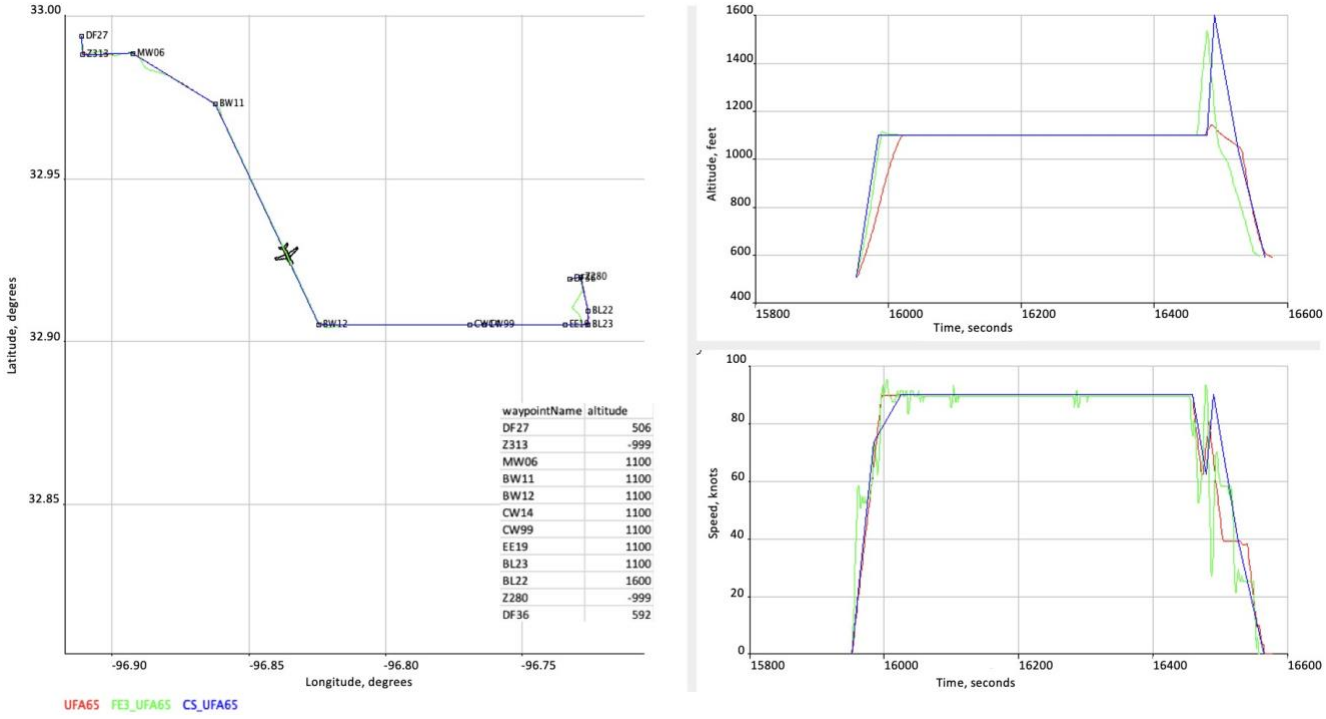



*Figure 15: UFA65 flight: MATG vs. Fe³ with 6DOF Model*

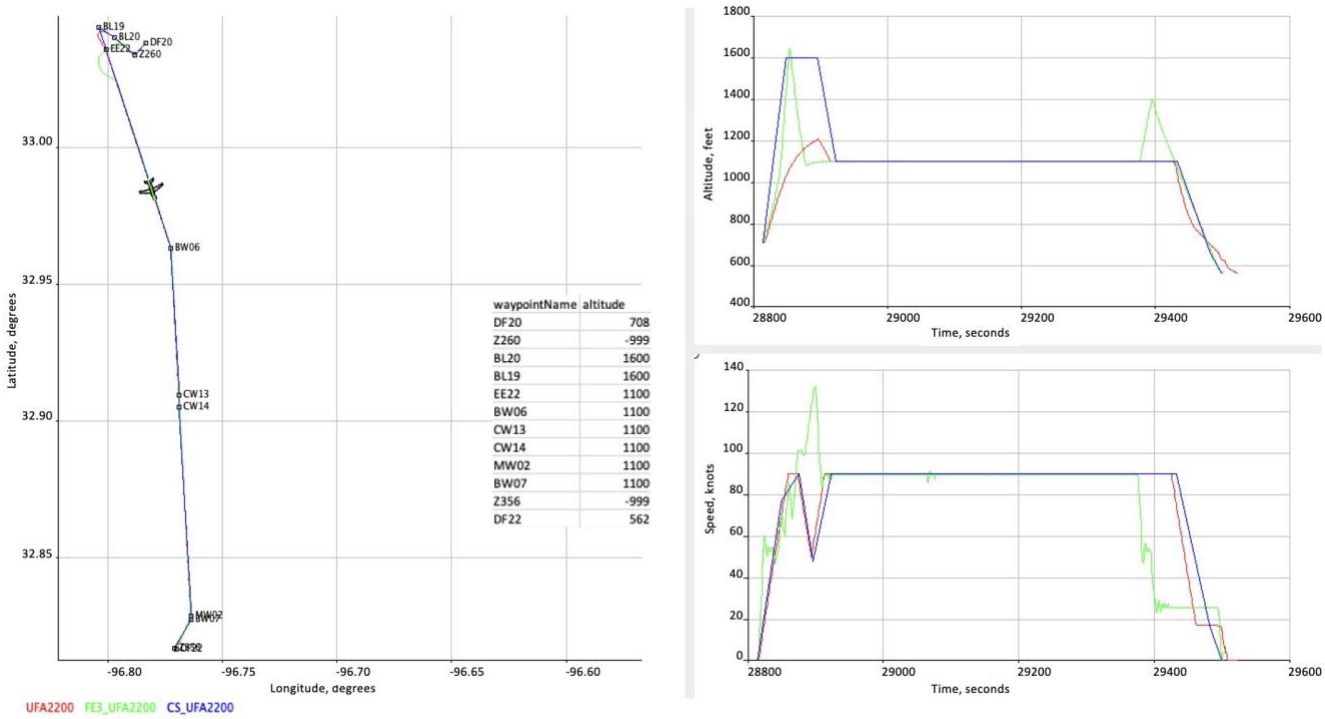



*Figure 16: UFA2200 flight: MATG vs. Fe³ with 6DOF Model*

Table 1 shows the comparison results for Fe³ with kinetic and 6DOF models, aggregated over eight test flights. The aggregated metrics are generated by the Automated Trajectory Comparison tool described in subsection 6.2.

*Table 1: Aggregated comparison results for Fe3*

| | |
|---|---|
| **Kinetic model:** | |
| Average mean flight separation<br>Average maximum flight separation<br>Average standard deviation of flight separation | 0.42 nmi<br>1.06 nmi<br>0.3791 |
| Average mean along track path difference<br>Average maximum along track path difference<br>Average minimum along track path difference<br>Average standard deviation of along track differences | -0.10 nmi<br>0.47 nmi<br>-0.62 nmi<br>0.3678 |
| Average mean time difference<br>Average maximum time difference<br>Average minimum time difference<br>Average standard deviation of time difference values | 24.14 sec<br>217.75 sec<br>-27.62 sec<br>62.8003 |
| **6DOF model:** | |
| Average mean flight separation<br>Average maximum flight separation<br>Average standard deviation of flight separation | 0.54 nmi<br>1.01 nmi<br>0.3197 |

| | |
|---|---|
| Average mean along track path difference<br>Average maximum along track path difference<br>Average minimum along track path difference<br>Average standard deviation of along track differences | 0.04 nmi<br>0.63 nmi<br>-0.39 nmi<br>0.3420 |
| Average mean time difference<br>Average maximum time difference<br>Average minimum time difference<br>Average standard deviation of time difference values | 4.07 sec<br>40.85 sec<br>-20.58 sec<br>17.0178 |

## 7.2 Comparisons with VFR track data

The previous section compared MATG-generated trajectories with trajectories generated by Fe[3] using a BADA-based "kinetic" model and a higher fidelity 6DOF quadrotor model. As valuable as these comparisons are, they involve only simulated trajectories and provide no proof that these trajectories are "realistic" and reflect a behavior of real aircraft. Such proof requires comparisons with real-world air traffic data from historical ground-based radar tracks available in the NASA Sherlock Data warehouse [17]. These comparisons, however, face methodological difficulties because recorded track data is inherently noisy and trajectories depend on pilot intent and aircraft performance data that may not be readily available, especially for VFR flights.

The following approach is used to overcome these difficulties. Using the C3FlightModeler tool, the intent and performance data is extracted from noisy track data for VFR flights as described in subsection 6.1. MATG uses this data to generate trajectories that can be compared against the recorded raw track data and the intent (flight plan) inferred from this data by C3FlightModeler cutting through the noise.

Figures 17 through 21 show results of some of these comparisons. Flight IDs in these figures are created by concatenation of Aircraft ID (call sign) and "flight Key", a unique integer assigned to a single flight. MATG-generated trajectories in these figures are shown in red. Track data are shown in green and indicated by the prefix "TR", and constraints representing the flight plans inferred from this data are shown in blue and indicated by the prefix "CS". These constraints were generated using *altitudeChangeLimit* set to 100 ft to remove altitude noise caused by transponders reporting pressure altitude information in 100-ft increments, and *turnRateLimit* set to 6 deg/sec, which is more suitable for modeling complex or irregular horizontal paths, such as in loitering patterns. The names of constraints in all the following plots are their sequential numbers.

Figure 17 shows comparison results for a flight along a straight line including a climb segment followed by a cruise segment. Track data for this flight has low altitude noise and a moderate level of speed noise that was effectively removed by C3FlightModeler. The track point after the last constraint was not included as a constraint because C3FlightModeler removed it as an outlier that would result in unreasonably high deceleration as described in subsection 6.1. The climb profile modeled by MATG generally follows the track data, but includes a short cruise segment near 10,000 ft. This is explained by applying a higher rate of climb from the initial climb to the whole flight segment, which is a limitation of aircraft performance modeling in C3FlightModeler rather than of MATG itself.

Figure 18 shows comparison results for a more interesting flight with a complex altitude and speed profile, ending with a loitering pattern. As can be seen in the altitude plot, C3FlightModeler effectively removes altitude noise caused by the 100-foot increments in recorded pressure altitude, and the MATGgenerated altitude closely follows this noise-free altitude profile. Speed profiles are also similar except for a segment between 21300 and 21400 seconds where both C3FlightModeler and MATG underestimated the speed, resulting in about a 30-second time delay. Figure 19 illustrates that in general, the MATG trajectory closely

follows the loitering pattern. It can be noted that the overestimated (doubled) bank angle as described in subsection 6.1 can result in deviations in sharp turns, such as between constraints 16 and 17, 44 and 45, or 84 and 85, but prevents skipping waypoints.

Figure 20 shows comparison results for another flight with a complex horizontal and vertical profile. In this case, C3FlightModeler slightly overestimates the cruise altitude by 100 ft, which is within the 100-ft tolerance, and cruise speed. At the same time, the MATG-generated altitude and speed profiles closely follow these altitude and speed constraints. As can be seen in Figure 21, the horizontal MATG-generated trajectory deviates most significantly from track data around constraints 7 through 9. This happens because of an initial misalignment between the course in the track data and the direction into constraint 7, leading to overestimated bank angle as was the case for the flight 0476 16495 shown in Figure 19. For the final turn segment between constraints 10 and 14, the MATG trajectory closely follows the track data.

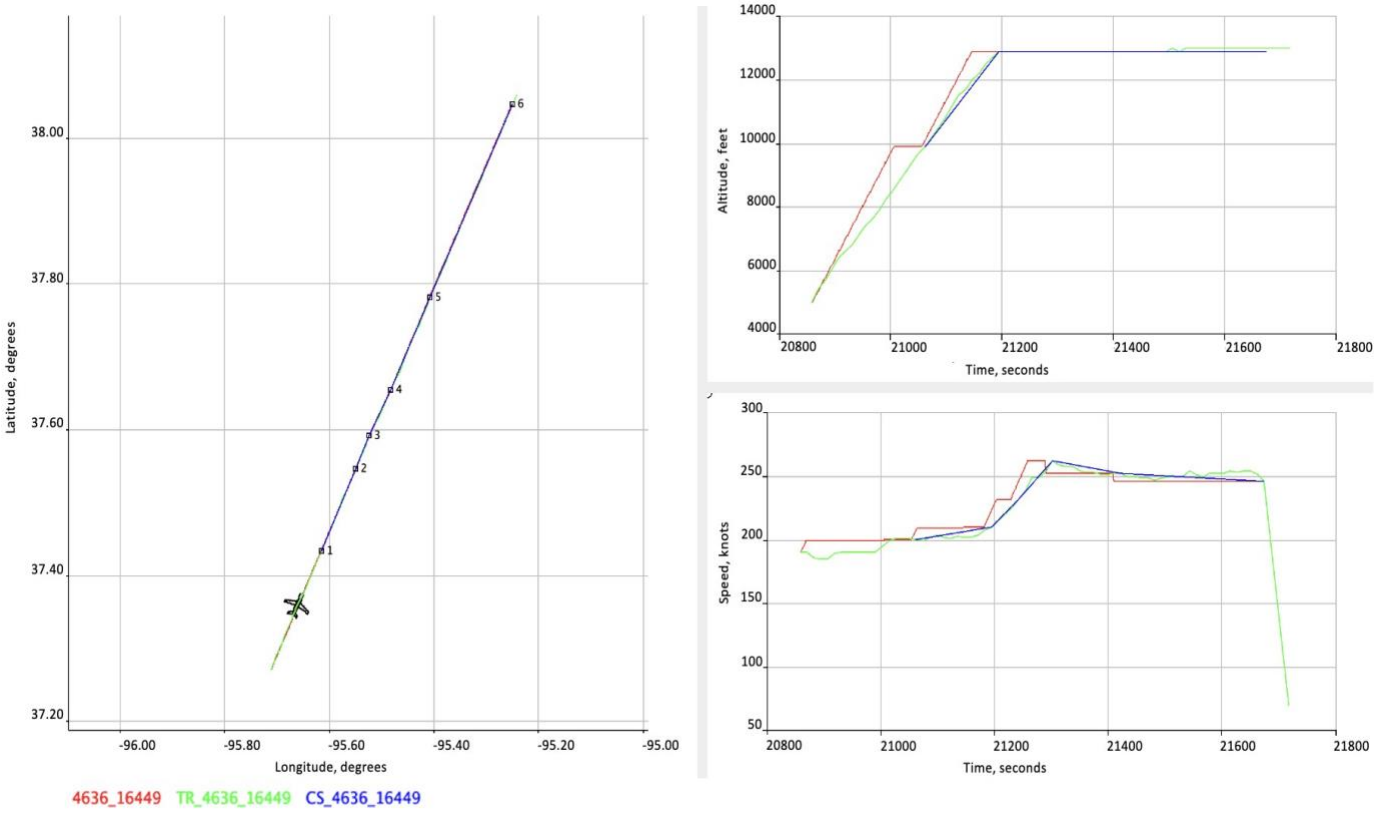



*Figure 17: 4636 16449 flight trajectory vs. track data*

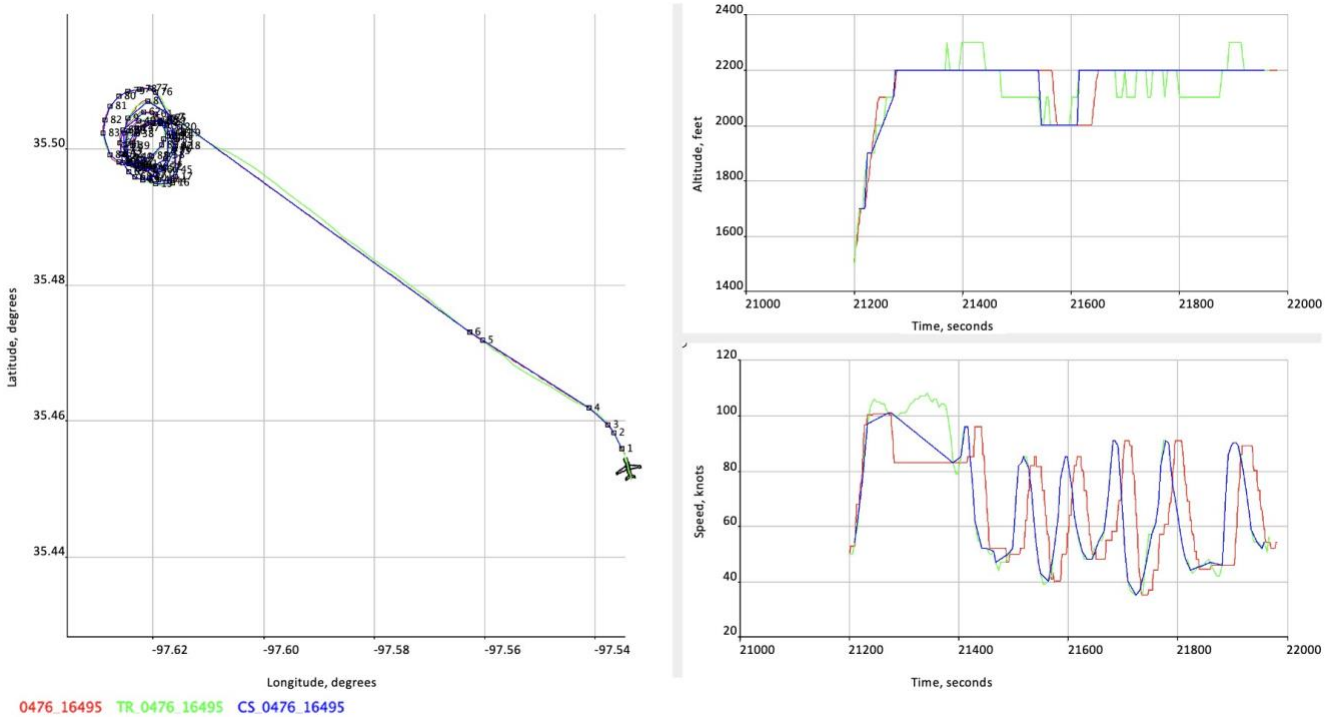



*Figure 18: 0476 16495 flight trajectory vs. track data*

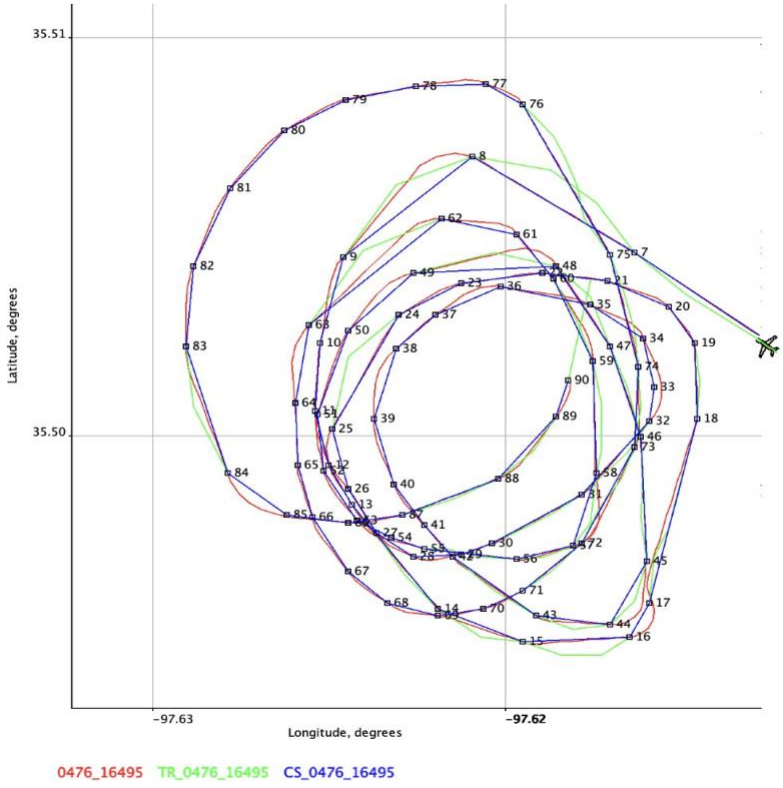



*Figure 19: 0476 16495 flight trajectory vs. track data: loitering pattern*

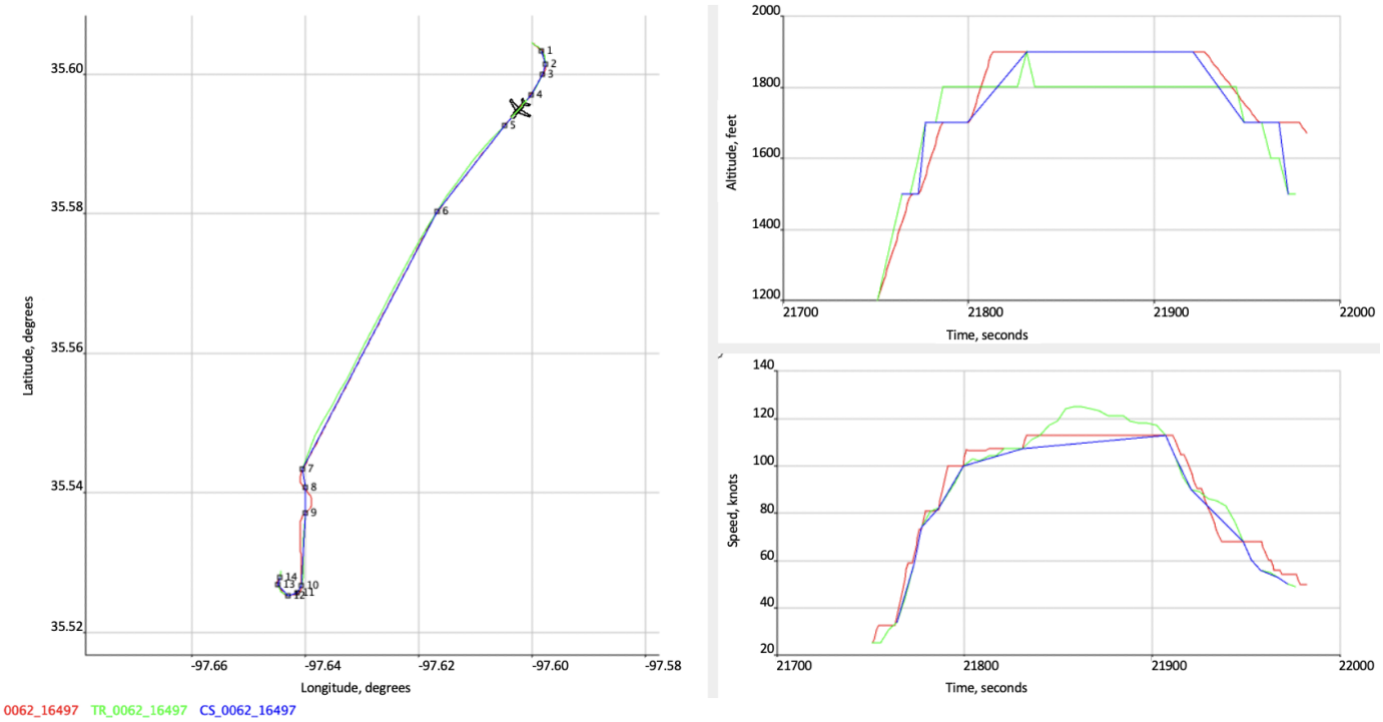



*Figure 20: 0062 16497 flight trajectory vs. track data*

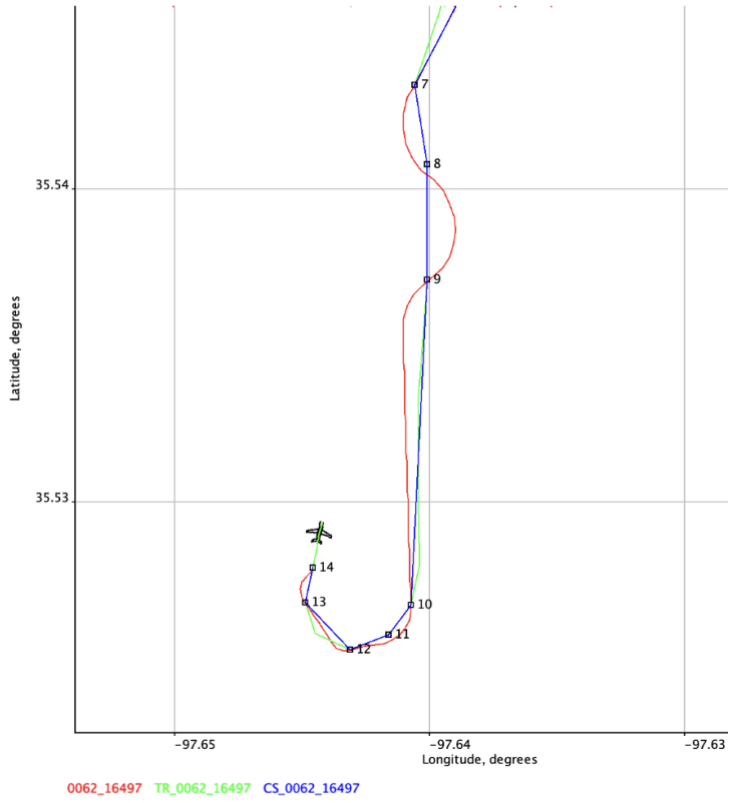



*Figure 21: 0062 16497 flight trajectory vs. track data: consecutive turns*

Table 2 shows the comparison results, generated by the Automated Trajectory Comparison tool, for VFR track data aggregated over six test flights. It can be concluded from these comparisons that MATG trajectories are reasonably close to recorded track data despite the noise in the data and uncertainties in intent and performance modeling. This indicates that MATG can generate realistic trajectories if accurate flight plans and APM are provided.

*Table 2: Aggregated comparison results for VFR track data*

| | |
|---|---|
| Average mean flight separation | 0.26 nmi |
| Average maximum flight separation | 0.75 nmi |
| Average standard deviation of flight separation | 0.2011 |
| Average mean along track path difference | 0.31 nmi |
| Average maximum along track path difference | 0.77 nmi |
| Average minimum along track path difference | -0.19 nmi |
| Average standard deviation of along track differences | 0.2750 |
| Average mean time difference | 3.88 sec |
| Average maximum time difference | 27.09 sec |
| Average minimum time difference | -7.02 sec |
| Average standard deviation of time difference values | 6.9092 |

## 7.3 Applications to ETA Trajectory Service

In some applications, nominal flight plans and APM don't fully define a trajectory because the aircraft behavior is guided by some additional rules. Such is the case when MATG is used by the ETAG to generate the ETAs at waypoints along a given route whenever Fleet Operator requests them in flight planning. The service works with a different flight simulation tool to generate the simulated traffic, the Airspace Traffic Generator (ATG) [1], which has its own speed control logic. Therefore, MATG needs to emulate the ATG's logic in order to generate accurate ETAs. This can be done using FCDL described in subsection 4.2.

The six commands listed in Table 3 were found sufficient to mimic the ATG speed control logic.

The first five commands are the "target-scope" commands executed in preprocessing and adjusting speeds for constraints specified by their *Conditions*. The "Target Type" in these commands refers to a prefix in waypoint name, e.g., all targets with names starting with "TF" have a type "TF." The last command, "Use cruise speed between waypoints," is the "state-scope" command that redefines the speeds only when it is executed by MATG. This command also uses the "dynamic variable" in its *Actions*. Its specific value is determined from the "plannedCruiseTas" property of each flight.

*Table 3: Flight commands for comparisons with ATG*

| | |
|---|---|
| **Reduce speed at sharp turns** | |
| *Conditions* | Target Course Change > 30 degrees |
| *Actions* | use Target TAS is 70 knots |
| **Limit departure and arrival speeds** | |
| *Conditions* | Target Type is TF |
| *Actions* | use Target TAS is 50 knots |
| **Reduce speed at Final Waypoints** | |
| *Conditions* | Target Type is FP |
| *Actions* | use Target TAS is 50 knots |
| **Adjust speed for final approach** | |
| *Conditions* | Target Type is G |
| *Actions* | use Target TAS is 50 knots |
| **Reduce speed at close sharpest turns** | |
| *Conditions* | Current Target Course Change > 45 degrees<br>Previous Distance to Target < 0.3 nmi |
| *Actions* | use Target TAS is 50 knots |
| **Use cruise speed between waypoints** | |
| *Conditions* | State Distance to Target > 1 nmi<br>Target Type is not G |
| *Actions* | use plannedCruiseTas knots |

The APM used by ATG needs to be approximated as well. In particular, the rates of climb and descent had to be defined as functions of altitude to match climb and descent profiles in trajectories generated by ATG.

Figures 22 and 23 show results of comparisons between MATG and ATG trajectories for two typical flights. The flight plans for these comparisons were based on scenarios for DFW airspace used in the Xseries simulations for the UAM Subproject of NASA's Air Traffic Management-eXploration project (ATM-X) [12]. As before, MATG-generated trajectories in these figures are shown in red. ATG-generated trajectories are shown in green and indicated by "ATG" prefix, and constraints representing the flight plans updated from flight commands in the preprocessing step are shown in blue and indicated by "CS" prefix.

As can be seen from the top views (left), the horizontal trajectories are practically indistinguishable between MATG and ATG and closely follow the given flight plans. The climb and descent profiles in the altitude plots are also close, except that ATG seems to use a little lower rate of final descent than the usually recommended 100 fpm used by MATG, resulting in slightly longer descents. The main results, illustrating the approximation of ATG speed control logic using flight commands in MATG, are reflected in the speed plots (bottom right). Despite significant variations in speeds during the flight, the red and green curves look similar. Therefore, the overall flight durations and the ETAs at intermediate points are also close.

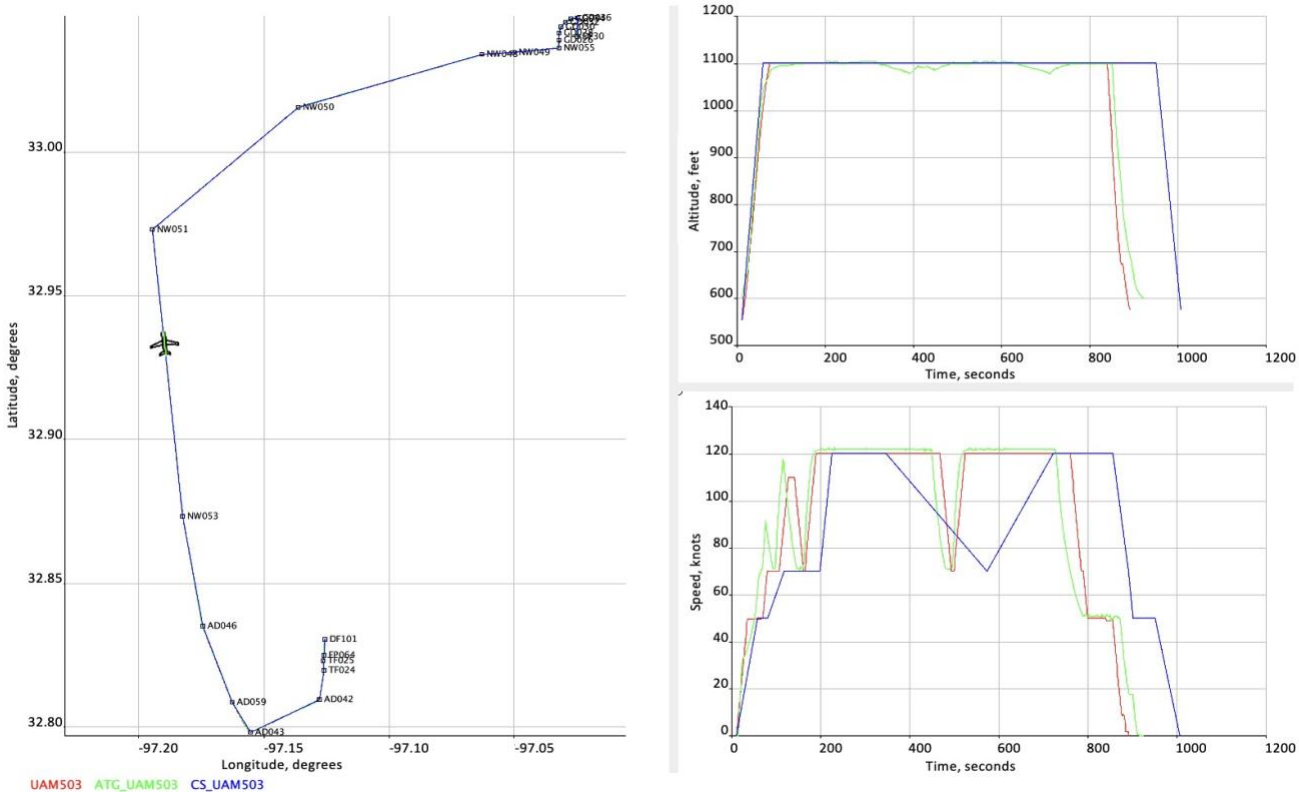



*Figure 22: UAM503 flight: MATG-generated trajectory vs. ATG-generated trajectory*

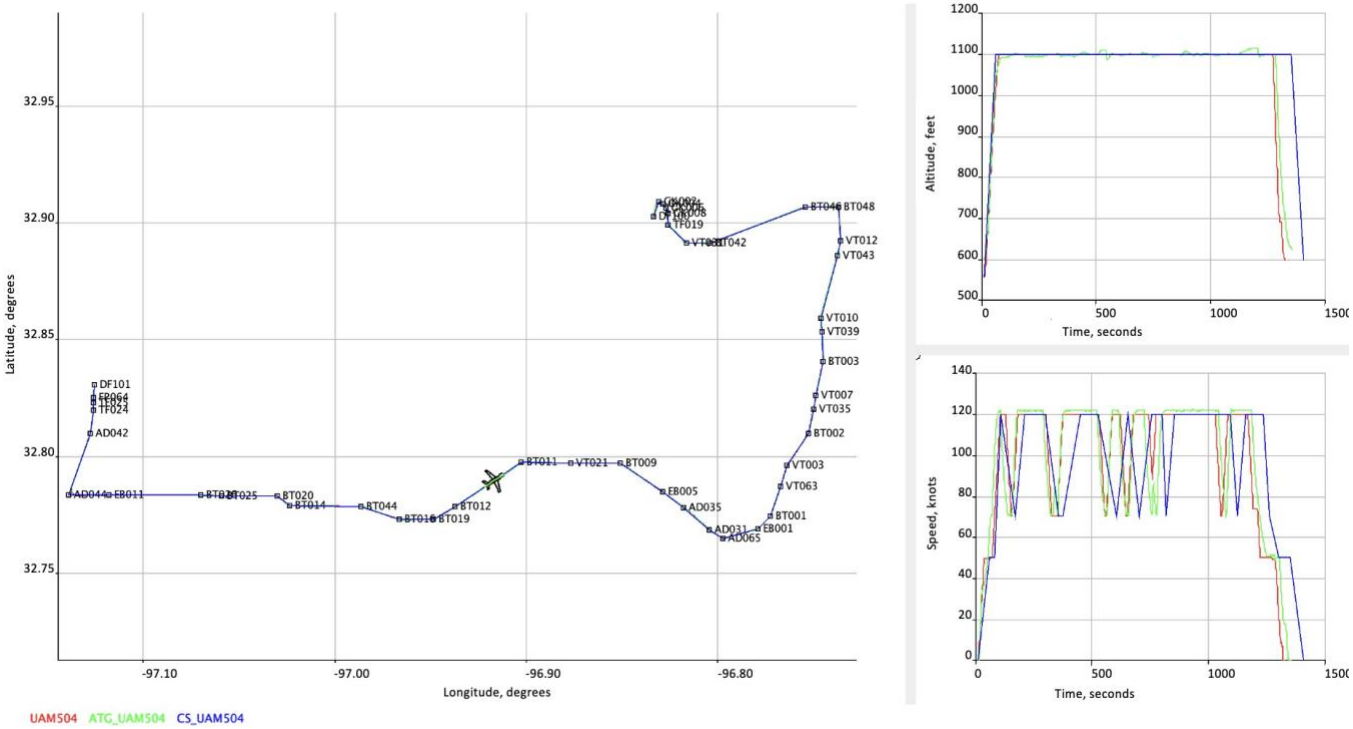



*Figure 23: UAM504 flight: MATG-generated trajectory vs. ATG-generated trajectory*

Table 4 shows the aggregated comparison results for 11 test flights generated by the Automated Trajectory Comparison tool. The table confirms the observations from Figures 22 and 23 that MATG and ATG trajectories are close for all test flights. What is most relevant for the ETAG service is that the mean time difference is only 7 sec, and even the average maximum time difference is well below the 1 min tolerance for ETA errors.

*Table 4: Aggregated comparison results for ATG trajectories*

| | |
|---|---|
| Average mean flight separation | 0.19 nmi |
| Average maximum flight separation | 0.44 nmi |
| Average standard deviation of flight separation | 0.1274 |
| Average mean along track path difference | 0.13 nmi |
| Average maximum along track path difference | 0.43 nmi |
| Average minimum along track path difference | -0.13 nmi |
| Average standard deviation of along track differences | 0.1540 |
| Average mean time difference | 7.18 sec |
| Average maximum time difference | 31.73 sec |
| Average minimum time difference | -3.94 sec |
| Average standard deviation of time difference values | 8.1924 |

# 1 Discussions and Summary

An emerging concept of UAM/AAM aims to provide a practical and cost-effective mobility alternative to ground transportation for the general public. Successful implementation of AAM operations is predicated upon maintaining or exceeding the level of safety and performance achieved by current operations in the NAS. Achieving these goals requires extensive research efforts that present unique challenges for modeling/simulation software tools. High traffic density and operational tempo, unusual airspace structures, different performance characteristics of eVTOL aircraft are just a few of many such challenges. Simulation software tools developed to support this research may need to generate realistic trajectories for high-density UAM traffic by orders of magnitude faster than real time. A fast kinematic Multimodal Adaptable Trajectory Generator (MATG) was created to provide this capability. It was originally developed as a part of the Java Architecture for DAA Extensibility and Modeling (JADEM), a fast-time simulation tool that was used to support research on alerting and guidance functions of UAS and UAM aircraft. Currently MATG is available as a standalone trajectory generator and as a library that can be called by other tools and services, such as the Estimated Times of Arrival (ETAs) Generation service (ETAG).

One of the keys to JADEM's versatility is the use of a flexible flight data model, expressing the flight plans in terms of *constraints* that can specify desired waypoints, heading, speed, and altitude. MATG also supports the Flight Commands Definition Language (FCDL) to model requirements or rules that cannot be expressed in the language of *constraints*. MATG can model diverse vehicle types, including non-traditional VTOL aircraft, by means of user-defined aircraft performance models (APM) that can specify bank angle or turn rate, horizontal and vertical accelerations, and altitude-dependent rates of climb and descent.

In addition, the tools for loading or inferring flight data from various data sources and for comparing flight trajectories were created. Two such tools, C3FlightModeler and FlightComparer, are described and used in this paper for the verification and validation of MATG.

An extensive experience of using MATG and comparisons with other flight modeling tools, such as Fe[3] and Airspace Traffic Generator (ATG), as well as with track data for real VFR flights confirmed the ability of MATG to generate accurate and realistic trajectories for diverse applications. For typical UAM flights with a 1-second time step, MATG can easily generate several thousands of trajectories in just a few seconds. This makes it suitable for modeling high density traffic and conflict management that typically requires generating multiple trial trajectories.

## References


1. *Airspace Traffic Generator: User's Manual Supplement, Revision 2.6.* SAIC, 2006.

2. User Manual for the Base of Aircraft DATA (BADA) Revision 3.8. Technical Report 2010-003, Eurocontrol Experimental Centre, 2010.

3. M. Abramson and K. Ali. Integrating the Base of Aircraft Data (BADA) in CTAS Trajectory Synthesizer. Technical Report NASA-TM-2012-216051, NASA Ames Research Center, Moffett Field, California, 2012.

4. M. Abramson, M. Refai, and C. Santiago. The Generic Resolution Advisor and Conflict Evaluator (GRACE) for Unmanned Aircraft Detect-And-Avoid Systems. Technical Report NASA/TM-2017219507, NASA Ames Research Center, Moffett Field, California, 2017.

5. J. A. Besada, G. Frontera, J. Crespo, E. Casado, and J. Lopez-Leones. Automated aircraft trajectory prediction based on formal intent-related language processing. In *IEEE Trans. Intell. Transp. Syst.*, volume 14, pages 1067–1082. IEEE, 2013.

6. K. Bilimoria, B. Sridhar, G. Chatterji, K. Sheth, and S. Grabbe. FACET: Future ATM Concepts Evaluation Tool. In *Air Traffic Control Quarterly*, volume 9, pages 1–20, 2001.

7. H. Bouadi and F. Mora-Camino. Modeling and Adaptive Flight Control for Quadrotor Trajectory Tracking. In *Journal of Aircraft*, volume 55, 2018.

8. G. Chatterji. Fuel Burn Estimation Using Real Track Data. In *AIAA-2011-6881, 11th American Institute of Aeronautics and Astronautics (AIAA) Aviation Technology, Integration, and Operations (ATIO) Conference*, Virginia Beach, VA, 2011. AIAA.

9. G. Chatterji. Trajectory Simulation for Air Traffic Management Employing a Multirotor Urban Air Mobility Aircraft Model. In *2020 AIAA Aviation Forum, Virtual Event*. AIAA, 2020.

10. G. Chatterji, B. Sridhar, and K. Bilimoria. En-route flight trajectory prediction for conflict avoidance and traffic management. In *AIAA Guidance, Navigation and Control Conference*, San Diego, CA, 1996. AIAA.

11. H. Erzberger, T. J. Davis, and S. M. Green. Design of Center-TRACON Automation System. In *Machine Intelligence in Air Traffic Management, edited by A. Benoit, AGARD CP-538*, page 11.1–11.12, 1993.

12. A. W. Cheng et al. National campaign (nc)-1 strategic conflict management simulation (x4) final report. Technical Report NASA/TM-2022-0018159, NASA, 2022. https://ntrs.nasa.gov/ citations/20220018159.

13. R. D. Falck, D. Ingraham, and E. Aretskin-Hariton. Multidisciplinary Optimization of Urban-Air Mobility Class Aircraft Trajectories with Acoustic Constraints. In *Proc. AIAA/IEEE Electric Aircraft Technologies Symposium*, Cincinnati, OH, 2018. IEEE/AIAA.

14. D. C. Hartman, C. L. Hartman, and J. V. Foster. Performance modeling of urban air mobility vehicles to support air traffic control studies. Technical report, Flight Dynamics Branch, NASA Langley Research Center, 2021.

15. A. Lee, M. G. Wu, and M. Abramson. Modeling of complex and diverse aircraft trajectories with the trajectory synthesizer generalized profile interface. In *AIAA Modeling and Simulation Technologies Conference*, Kissimmee, FL, 2015. AIAA.

16. H. Lee, A. Lee, L. Guillermo, C. Seah, and K. Moolchandani. Simulated evaluation of strategic conflict management capabilities for urban air mobility operations. In *AIAA Aviation 2024 Forum*, Las Vegas, NV, 2024. AIAA.

17. S. Lee, M. Abramson, J. D. Phillips, and H. Tang. Preliminary Analysis of Separation Standards for Urban Air Mobility using Unmitigated Fast-Time Simulation. In *41st Digital Avionics Systems Conference (DASC)*, Portsmouth, VA, 2022.

18. J. Lopez-Leones, M. A. Vilaplana, E. Gallo, F. A. Navarro, and C. Querejeta. The Aircraft Intent Description Language: A key enabler for air-ground synchronization in Trajectory-Based Operations. In *IEEE/AIAA 26th Digital Avionics Systems Conference*, Dallas, TX, 2007. IEEE.

19. L. Meyn, R. Windhorst, K. Roth, D. V. Drei, G. Kubat, V. Manikonda, S. Roney, G. Hunter, A. Huang, and G. Couluris. Build 4 of the Airspace Concept Evaluation System. In *Proc. AIAA Modeling and Simulation Technologies Conference and Exhibit*, Keystone, Colorado, 2006. AIAA.

20. M. D. Patterson, K. R. Antcliff, and L. W. Kohlman. A Proposed Approach to Studying Urban Air Mobility Missions Including an Initial Exploration of Mission Requirements. In *Proc. American Helicopter Society International 74th Annual Forum & Technology Display*, Phoenix, AZ, 2018.

21. M. Rimjha, M. Li, N. Hinze, S. Tarafdar, S. Hotle, H. Swingle, A. Trani, and J. C. Smith. Demand forecast model development and scenarios generation for urban air mobility concepts. Technical report, Virginia Polytechnic Institute University, 2020.

22. R. Slattery and Y. Zhao. Trajectory Synthesis for Air Traffic Automation. In *Journal of Guidance, Control and Dynamics*, volume 20, pages 232–238, 1997.

23. H. Tang, S. Lee, M. Abramson, and J. D. Phillips. Analysis of Conflicts among Urban Air Mobility Aircraft and with Traditional Aircraft. In *41st Digital Avionics Systems Conference (DASC)*, Portsmouth, VA, 2022.

24. M. Xue, J. Rios, J. Silva, Z. Zhu, and A. K. Ishihara. Fe$^3$: An evaluation tool for low-altitude air traffic operations. In *Proceedings of the 2018 Aviation Technology, Integration, and Operations Conference*, pages 1–13, Atlanta, Georgia, 2018.

25. Y. Zhang, G. Satapathy, V. Manikonda, and N. Nigam. KTG: A Fast-Time Kinematic Trajectory Generator for Modeling and Simulation of ATM Automation Concepts and NAS-wide System Level Analysis. In *Proc. AIAA Modeling and Simulation Technologies Conference*, Toronto, Ontario, Canada, 2010. AIAA.