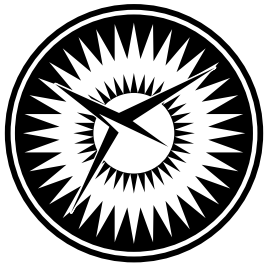
NASA/CR-2007-214863
NIA Report No. 2007-06

NATIONAL
INSTITUTE OF
AEROSPACE



In-Trail Procedure (ITP) Algorithm Design

Cesar A. Munoz and Radu I. Siminiceanu
National Institute of Aerospace (NIA), Hampton, Virginia

August 2007

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published 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 having 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 that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results ... even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

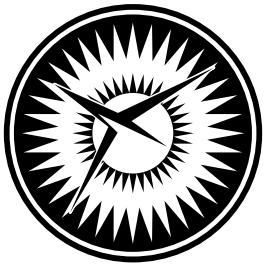
- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Phone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/CR-2007-214863
NIA Report No. 2007-06

NATIONAL
INSTITUTE OF
AEROSPACE



In-Trail Procedure (ITP) Algorithm Design

Cesar A. Munoz and Radu I. Siminiceanu
National Institute of Aerospace (NIA), Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Cooperative Agreement NCC1-02043

August 2007

Available from:

NASA Center for AeroSpace Information (CASI)
7115 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

IN-TRAIL PROCEDURE (ITP) ALGORITHM DESIGN*

César A. Muñoz and Radu I. Siminiceanu †

ABSTRACT

The primary objective of this document is to provide a detailed description of the In-Trail Procedure (ITP) algorithm, which is part of the Airborne Traffic Situational Awareness – In-Trail Procedure (ATSA-ITP) application. To this end, the document presents a high level description of the ITP Algorithm and a prototype implementation of this algorithm in the programming language C.

Acronyms	
ADS-B	Automatic Dependent Surveillance-Broadcast
API	Application Programming Interface
ATSA	Airborne Traffic Situational Awareness
ITP	In-Trail Procedure
FIR	Flight Information Region
NATOTS	North Atlantic Organized Track System
RVSM	Reduced Vertical Separation Minimum
TCAS	Traffic Collision Avoidance System

1 INTRODUCTION

The In-Trail Procedure (ITP) algorithm is an airborne software module of the Airborne Traffic Situational Awareness – In-Trail Procedure (ATSA-ITP) application in non-radar airspace. This software module is responsible for computing the flight deck information needed by the crew to determine whether the criteria required for an ITP procedure are met or not. This information is processed by a host application that decides how and when it is displayed to the crew.

This document is organized as follows. Section 2 presents a list of definitions that are relevant to the ITP algorithm. Section 3 gives a high level description of the algorithm that includes assumptions, configurable parameters, input and output data, and functional behavior. Finally, a prototype implementation of the algorithm is described in Section 4. Appendix A lists the complete code of the prototype implementation. Appendix B includes auxiliary functions from Williams' Aviation Formulary [3].

2 DEFINITIONS

The following is a list of definitions, based on the list provided in [2], that are relevant to the description of the ITP algorithm.

- **Altitude**. "The vertical distance of a level, a point or an object considered as a point, measured from mean sea level (Doc 4444 - PANS-ATM chapter 1)." [2].

*This work was supported in part by the National Aeronautics and Space Administration under the cooperative agreement NCC-1-02043.

†National Institute of Aerospace, 100 Exploration Way, Hampton VA, 23666. E-mail: {munoz,radu}@nianet.org. Authors are in alphabetical order.

- **Flight Level**. "A surface of constant atmospheric pressure which is related to a specific pressure datum, and is separated from other such surfaces by specific pressure intervals (Doc 4444 - PANS-ATM chapter 1)." [2]. On a day with standard atmospheric pressure (defined in the 1976 US Standard Atmosphere as 29.92 inches of mercury), a flight level of 310 would correspond to an altitude of 31,000 feet mean sea level. Flight levels are always in increments of 1,000 feet.

- **Flight Level Dead Band**. Maximum altitude deviation beyond which an aircraft is not considered to be in a flight level.

- **Ground Speed Differential**. "The speed difference over the ground between the ITP Aircraft and the Potentially Blocking Aircraft along each aircraft's track. This measurement would use a technique similar to the Doc 4444 - PANS-ATM longitudinal separation procedure using DME (Doc 4444 - PANS-ATM section 5.4.2.3) to determine the Ground Speed Differential. A Positive Ground Speed Differential signifies that the ITP Aircraft and the Reference Aircraft are closing on each other (the separation is being reduced)." [2].

- **Initial Flight Level**. "The flight level of the ITP Aircraft when it determines a climb or descent is desired and before it begins the climb or descent." [2].

- **In Trail Procedure**. "A procedure employed by an aircraft that desires to change its flight level to a new flight level by climbing or descending in front or behind one, or between two Same Track, Potentially Blocking Aircraft which are at an Intervening Flight Level and which are at less than the standard longitudinal separation minimum." [2].

- **ITP Aircraft**. "An aircraft that is fully qualified (from an equipment, operator, and flight crew qualification standpoint) to conduct an ITP and whose flight crew is considering a change of flight level." [2]. For the ITP algorithm, the ITP Aircraft is assumed not to be maneuvering.

- **ITP Maximum Range**. Maximum horizontal range relative to ownship for any aircraft to be included in the traffic list.

- **Maneuvering Aircraft**. Nearby Aircraft whose altitude is between the ITP Aircraft's initial flight level and the Requested Flight Level, but off the altitude dead bands.

- **Nearby Aircraft**. Aircraft within ITP Maximum Range that are flying in the Same Vicinity, in the Same or Opposite Direction (only aircraft flying in a Cross Direction are excluded).

- **Nearest Aircraft**. At most two Potentially Blocking Aircraft that are closest to the ITP Aircraft. For the ITP algorithm, if the two nearest aircraft are flying in the same direction and at the same flight level, only the closest aircraft is considered to be a nearest aircraft.

- **Opposite Direction**. Aircraft are flying in Opposite Direction if their relative bearing is between 135 degrees and 225 degrees.

- **Positive Mach Difference**. "The difference in Mach between the ITP Aircraft and the Reference Aircraft that would result in the aircraft closing on each other (the separation is being reduced)." [2].

- **Potentially Blocking Aircraft**. "Aircraft flying in the Same Direction at an Intervening Flight Level whose ADS-B report data are available to the ITP Aircraft." [2]. For the ITP algorithm, aircraft data of a Potentially Blocking Aircraft may not be qualified.

- **Qualified ADS-B**. "Automatic Dependent Surveillance-Broadcast (ADS-B) that meets the accuracy and integrity requirements determined to be required for the ITP." [2].

- **Reference Aircraft**. "One or two Potentially Blocking Aircraft that meet the ITP criteria and that will be identified to ATC by the ITP Aircraft as part of the ITP clearance request." [2]. For the ITP algorithm, the data quality of a Reference Aircraft must be qualified for ITP. Furthermore, a Reference Aircraft is not maneuvering and is either the closest ahead or the closest behind Potentially Blocking Aircraft.

- **Requested Flight Level**. "One same direction flight level above (for a climb) or below (for a descent) the Intervening Flight Level, appropriate for the operational region. A requested flight level may be:

  a. 2,000 feet above or below the initial flight level in RVSM airspace with single-direction tracks.

  b. 4,000 feet above or below the initial flight level in non-RVSM airspace with single-direction tracks.

  c. 4,000 feet above or below the initial flight level in RVSM airspace with bi-directional tracks with opposite direction traffic at alternating flight levels." [2].

- **Same Direction**. Aircraft are flying in the Same Direction if their relative bearing is within ± 45 degrees.

- **Same Vicinity**. Aircraft are located in the Same Vicinity, if their lateral offset is less than a Maximum Lateral Offset.

- **Same Track**. "Same direction tracks and intersecting tracks or portions thereof, the angular difference of which is less than 45 degrees or more than 315 degrees, and whose protection areas overlap (i.e., without lateral separation). (Doc 4444 - PANS-ATM section 5.4.2.1.5). These tracks may be a portion of a user preferred route or may be published routes, either fixed or flexible." [2]. Since the information necessary to compute the same track criterion is not available to the aircraft, the ITP algorithm uses instead the more general concept of Same Direction, Same Vicinity.

- **Track**. "The position on the earth's surface of the path of an aircraft, the direction of which path at any point is usually expressed as degrees from North (Doc 4444 - PANS-ATM Chapter 1)." [2].

- **Vertical Speed Dead Band**. Maximum vertical speed deviation beyond which an aircraft is considered to be maneuvering.

3 HIGH LEVEL DESCRIPTION

The main function of the ITP algorithm is called `itpAlgorithm`. This function is expected to run at a higher frequency than the traffic aircraft information is received. Hence, `itpAlgorithm` assumes that the input data have been coasted and that error-checking has been performed prior to its execution. More precisely, we assume:

- The position of each aircraft is projected to current time from the last state vector and elapsed time. The error between the real position and the projected one is bounded by the maximum cruise speed multiplied by the frequency of the algorithm.

- Traffic input data have an indication of their quality. The value `itpQualified` indicates that the aircraft is equipped with an ADS-B that meets the accuracy and integrity required to perform an ITP. In any other case, including the case of aircraft only equipped with TCAS, this value is set to `itpNonQualified`. Aircraft data that is `itpQualified` may be used for the computation of potentially blocking and reference aircraft. Aircraft data that is `itpNonQualified` may only be used for the computation of potentially blocking aircraft. In any case, for the purpose of the calculations, the data is supposed to be correct.

- The ADS-B reception range is much larger than the separation standards.

3.1 Run-time Configurable Parameters

The run-time configurable parameters of `itpAlgorithm` are depicted in Figure 1. A data structure `itpGlobalParamaters` includes the following parameters:

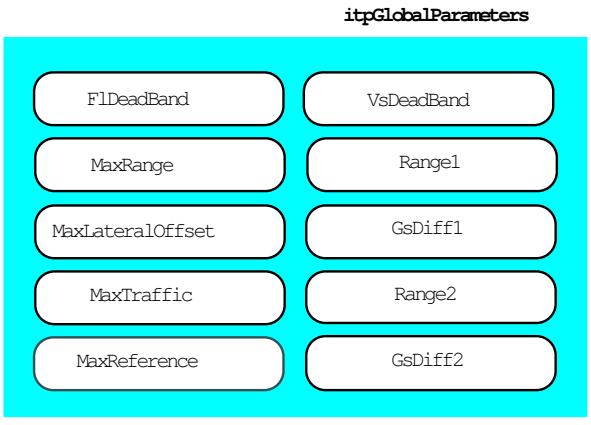


Figure 1: *Run-time configurable parameters of ITP algorithm*

- `FlDeadband`. Value of *Flight Level Dead Band* in hundreds of feet. The default value is 1.

- `VsDeadband`. Value of *Vertical Speed Dead Band* in feet per minute. The default value is 300.
- `MaxRange`. Value of *ITP Maximum Range* in nautical miles. Currently, the default value of this parameter is set to 160 (see below).
- `MaxLateralOffset`. Maximum lateral offset, in nautical miles, used for the definition of *Same Vicinity*. Since the official Lateral Separation is defined by the current Flight Information Region (FIR) where the aircraft is located and since this information may not be available to the crew, further analysis is needed to determine an appropriate value.
- `Range1` and `Range2`. The values of the distance parameters used in determining the ITP initiation criteria, in nautical miles. The default values are set to 15 and 20 nautical miles, respectively.
- `GsDiff1` and `GsDiff2`. The values of the ground speed differential parameters used in determining the the ITP initiation criteria, in knots. The default values are set to 20 and 30 knots, respectively.

The `itpMaxRange` configurable parameter is the maximum range, in nautical miles, beyond which requesting an ITP would not make sense. It is calculated using the Mach number technique chart for a distance to fly between 1 and 600 nautical miles (in most cases the distance between two consecutive way points should not exceed 600 nautical miles) and a Mach difference of 0.03 (as specified for reference aircraft in [2]). In this case, the chart gives an additional 3 minutes separation at common point to ensure at least minimum separation at exit point. Therefore, a 15 minutes required separation at exit point (NATOTS uses 10 minutes but 15 minutes may be used in some airspaces [1]) amounts to a 18 minutes separation at common point. Assuming a maximum cruising speed of Mach 0.87 (based on NATOTS data – see [2]) and a 29000 feet altitude (which is the lowest NATOTS altitude [1]), the maximum cruising speed is 514.93 knots (no wind is assumed). This yields a maximum separation for an ITP of $514.93 \times 18/60 \approx 154.48$ nautical miles, which is rounded up to 160 nautical miles.

There are also two **non**-run-time configurable parameters:

- `ITP_MAX_TRAFFIC`. Maximum size of traffic aircraft list. This limit is imposed by the design of the avionics system. Currently, this parameter is set to 159, as actual systems allow storage of data for up to 128 traffic aircraft equipped with ADS-B and up to 31 traffic aircraft equipped with TCAS only (aircraft that have both ADS-B and TCAS count only as ADS-B.)
- `ITP_MAX_REFERENCE`. Maximum number of aircraft that may meet the definition of *Reference Aircraft*. Currently, this parameter is set to 2, corresponding to the nearest potentially blocking aircraft in front and behind ownship, if either of those exist. This setting corresponds to the case of a single Intervening Flight Level. The subsequent use of this algorithm for multiple Intervening Flight Levels would require the adjustment of this parameter according to the corresponding concept of operations.

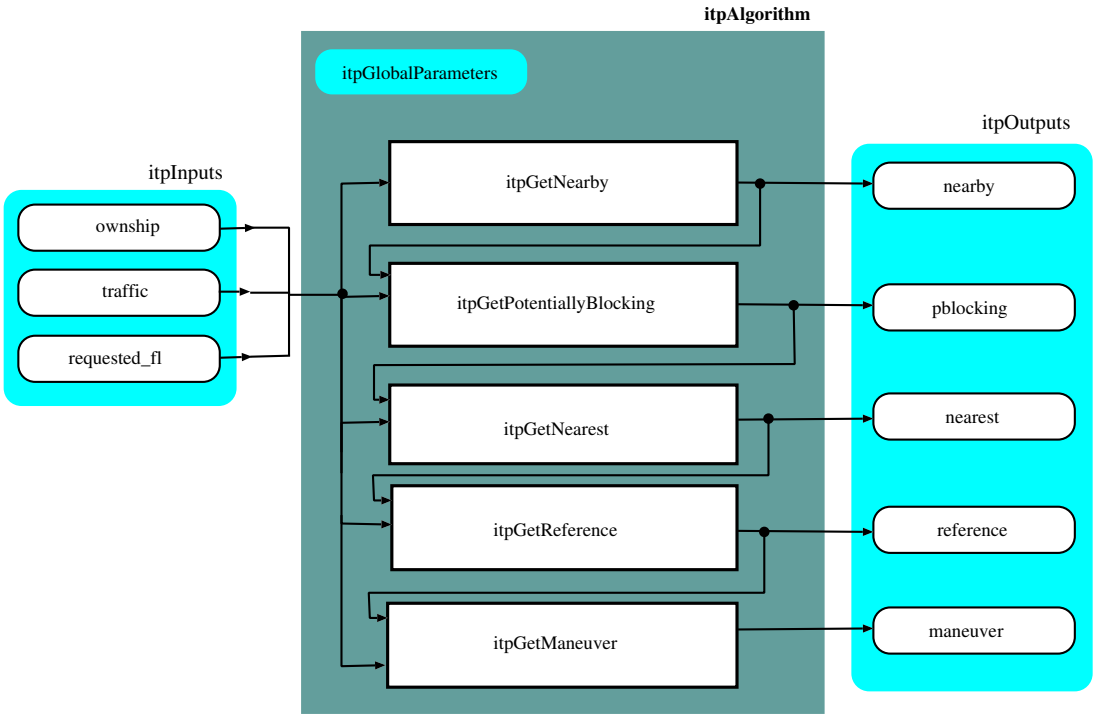


Figure 2: *Data flow chart of ITP algorithm*

3.2 Data Flow

Figure 2 presents a data flow chart of `itpAlgorithm`.

3.2.1 Inputs

The function `itpAlgorithm` gets as inputs a structure `itpInputs` consisting of:

- `ownship`. Data of ownship aircraft, of type `itpAircraft`. This aircraft should meet the definition of *ITP Aircraft*.
- `traffic`. A list of coasted and error-checked traffic aircraft data. The maximum length of this list is given by the non-configurable parameter `ITP_MAX_TRAFFIC`. The elements of this list are also of type `itpAircraft`.
- `requested_fl`. The requested cruise flight level (in hundreds of feet).

The `itpAircraft` data structure consists of

- a unique *flight identifier* (`id`), of type `itpIdentifier` (strings of up to 255 characters),
- *latitude* (`lat`), in degrees, in the interval $[-90^o, 90^o]$,
- *longitude* (`lon`), in degrees, in the interval $[-180^o, 180^o)$ – excluding 180^o; by convention East longitudes are positive,

- *altitude* (`alt`) at standard barometric pressure, in hundreds of feet,
- *ground speed* (`gs`), in knots,
- *vertical speed* (`vs`), in feet per minute,
- *ground track* (`trk`), in degrees from true north, in the interval $[0, 360)$ – excluding 360^o, and
- *data quality flag* (`qlty`), which is either `itpQualified` or `itpNonQualified`.

3.2.2 Outputs

The function `itpAlgorithm` outputs a data structure `itpOutput` that consists of a number of lists of `itpTraffic` elements:

- `traffic`: a list of all aircraft that are within the `itpMaxRange`.
- `nearby`: a list of indices to `traffic` of all aircraft that meet the definition of *Nearby Aircraft*.
- `pblocking`: a list of indices to `traffic` of all aircraft that meet the definition of *Potentially Blocking Aircraft*. The maximum length of this list is given by the non-configurable parameter `ITP_MAX_TRAFFIC`.
- `nearest`: a list of indices to `traffic` of all aircraft that meet the definition of *Nearest Aircraft*.
- `reference`: a list of indices to `traffic` of aircraft that meet the definition of *Reference Aircraft*. The maximum length of this list is also `ITP_MAX_REFERENCE`.
- `maneuver`: the type of In-Trail Procedure that is available, one of : {`itpNone, itpFollowingClimb, itpFollowingDescent, itpLeadingClimb, itpLeadingDescent, itpCombinedLeadingFollowingClimb, itpCombinedLeadingFollowingDescent`}.

The data structure `itpTraffic` consists of the following fields, which are computed by `itpAlgorithm`:

- *ITP range* (`range`), in nautical miles: the relative distance to ITP aircraft with respect to a common point. A negative value means that the aircraft is behind the ITP aircraft. Conversely, a positive value means that the aircraft is ahead of the ITP aircraft.
- *ground speed differential* (`gsdiff`), in knots: the relative ground speed differential to ITP aircraft. A negative value means that the aircraft is slower than the ITP aircraft. Conversely, a positive value means that the aircraft is faster than the ITP aircraft.
- *lateral offset* (`offset`): traffic relative lateral offset to ownship, in nautical miles.

- *relative position* (`relpos`): traffic relative position to ownship, from the enumerated type `itpRelPos`: {`itpOwnBehindTraffic, itpOwnAheadTraffic, itpNoRelPos`}, where `itpOwnBehindTraffic` is assigned to traffic leading ownship, `itpOwnAheadTraffic` to traffic following ownship, and `itpNoRelPos` to traffic that is flying in a cross direction, i.e., traffic that is neither same direction nor opposite direction.

- *direction of flight* (`dir`), from the enumerated type `itpDirection`: {`itpSameDirection, itpOppositeDirection, itpCrossDirection`}.

- *traffic type* (`actype`), a numeric value that encodes the role of the traffic aircraft in the ITP and, when necessary, an ITP error code corresponding to this aircraft. This numerical code is computed by binary addition on the following constants (in hexadecimal):

  - `ITP_TRAFFIC 0x0`
  - `ITP_NEARBY 0x1`
  - `ITP_POTENTIALLY_BLOCKING 0x2`
  - `ITP_NEAREST 0x4`
  - `ITP_REFERENCE 0x8`
  - `ITP_BLOCKING 0x10`
  - `ITP_MANEUVERING 0x20`
  - `ITP_TOO_FAST 0x40`
  - `ITP_TOO_CLOSE 0x80`
  - `ITP_UNQUALIFIED_DATA 0x100`

  For example, a nearby aircraft that is maneuvering has the associated `actype` value of `0x1` + `0x20` = `0x21` hexadecimal (that is 33 decimal value).

3.3 Control Flow

As illustrated in Figure 3, the function `itpAlgorithm` calls, in sequence, the following functions.

- `itpGetNearby`. This function computes the list `nearby`. If the list is empty the execution of `itpAlgorithm` ends.

- `itpGetPotentially_Blocking`. This function computes the list `potentially_blocking`. If the list is empty, the execution of `itpAlgorithm` ends.

- `itpGetReference`. This function computes the list `reference`.

- `itpGetManeuver`. Given the list `reference`, this function computes the ITP maneuver that is available to the ITP aircraft (or `itpNone`).

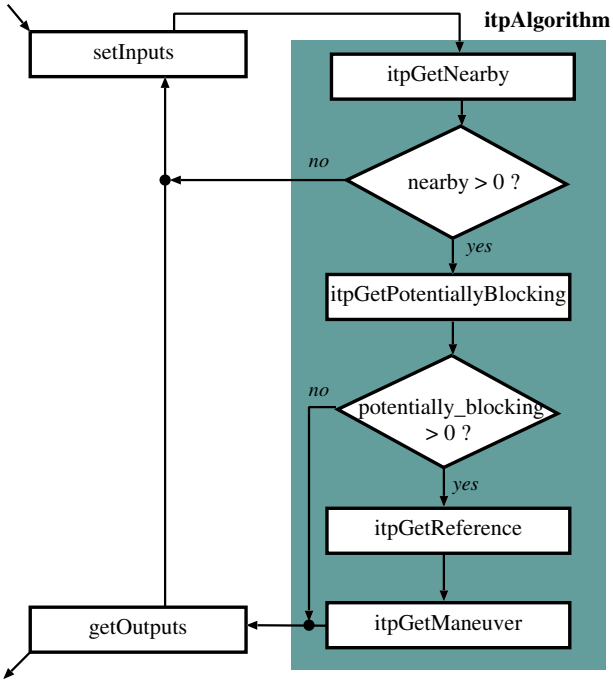


Figure 3: *Control flow chart of ITP algorithm*

4 PROTOTYPE IMPLEMENTATION

A prototype implementation of the ITP Algorithm, written in the programming language C, is electronically available at `http://research.nianet.org/~munoz/ITP_Module`. The implementation consists of the following files:

- `ITP_Defs.h`, included in Appendix A.1, contains constant and type declarations.
- `ITP_Parameters.c`, included in Appendix A.2, contains the definition of configurable parameters and utility functions to read and modify their values.
- `ITP_Algorithm.h`, included in Appendix A.3, contains the program interface.
- `ITP_Algorithm.c`, included in Appendix A.4, contains the definition of the main functions.
- `ITP_Aviation.h` and `ITP_Aviation.c`, included in Appendix B, contain the implementation of auxiliary functions from Williams' Aviation Formulary [3].

In the implementation, we use the following naming convention:

- Constants are written in uppercase letters and start with the prefix `ITP_`.
- Global variables, types, and function names start with the prefix `itp`.

The actual In-Trail Procedure is implemented by the function `itpAlgorithm`, which has the following interface:

```
void itpAlgorithm(itpInputs, itpOutputs*);
```

The function `itpAlgorithm` has two parameters: an input parameter of type `itpInputs`, which is passed by value, and an output parameter of type `itpOutputs`, which is passed by reference. The exact definition of these types is found in the file `ITP_Defs.h`.

The configurable parameters of the algorithm are defined as global variables in the file `ITP_Parameters.c`. These variables are not intended to be directly read or modified by the user. For that reason, the following functions are provided in the interface:

```
itpAltitude itpGetGlobalFlDeadband();
double      itpGetGlobalVsDeadband();
double      itpGetGlobalMaxRange();
double      itpGetGlobalMaxLateralOffset();
double      itpGetGlobalRange1();
double      itpGetGlobalGsDiff1();
double      itpGetGlobalRange2();
double      itpGetGlobalGsDiff2();

void itpSetGlobalFlDeadband(itpAltitude);
void itpSetGlobalVsDeadband(double);
void itpSetGlobalMaxRange(double);
void itpSetGlobalMaxLateralOffset(double);
void itpSetGlobalRange1(double);
void itpSetGlobalGsDiff1(double);
void itpSetGlobalRange2(double);
void itpSetGlobalGsDiff2(double);

void itpResetGlobalParameters();
```

As expected, functions with the prefix `itpGetGlobal` read the variables and functions with the prefix `itpSetGlobal` write them. The function `itpResetGlobalParameters` resets all the configurable global parameters to their default value.

Finally, the following functions are provided to check and decode the information returned by the ITP algorithm:

```
itpBool itpIsClosing(itpTraffic trf);
itpBool itpIsNearby(itpTraffic trf);
itpBool itpIsPotentiallyBlocking(itpTraffic trf);
itpBool itpIsNearest(itpTraffic trf);
itpBool itpIsReference(itpTraffic trf);
itpBool itpIsBlocking(itpTraffic trf);
itpBool itpIsManeuvering(itpTraffic trf);
itpBool itpIsUnqualifiedData(itpTraffic trf);
itpBool itpIsTooClose(itpTraffic trf);
itpBool itpIsTooFast(itpTraffic trf);
```

```
void itpBlckCode2String(itpIdentifier s,itpTraffic trf);
```


ACKNOWLEDGMENTS

The groundwork for the ITP module design was laid by the NASA Enhanced Oceanic Operations research group. Special credit goes to Mike Palmer, Frank Bussink, Ryan Chartrand, James Chamberlain, and Jeffrey Maddalon for design ideas, discussions, and suggestions that shaped up this ITP software module. The authors are also thankful to Ed Williams for his technical help with some of the formulas in his Aviation Formulary.

REFERENCES

[1] ICAO PANS-ATM. Doc. 4444, 2001.

[2] RTCA/EUROCAE Requirements Focus Group, Application Definition Sub-group. OSED In-Trail Procedure in Non-Radar Oceanic Airspace, version 5.0, 2006. Working draft.

[3] E. Williams. Aviation formulary v1.42. `http://williams.best.vwh.net/avform.htm`.

A IMPLEMENTATION OF THE ITP ALGORITHM

A.1 Constant and Type Declaration

```
/*=========================================================
  ITP algorithm constants and types
  Author: Cesar A. Munoz (NIA)
          Radu I. Siminiceanu (NIA)

  File   : ITP_Defs.h
  Release: ITP_Module-6.d (10/11/06)
  ========================================================*/

#ifndef ITP_DEFS
#define ITP_DEFS

/* General constants */
#define ITP_STR_LEN          255
#define ITP_VER              "ITP_Module-6.d (10/11/06)"
#define ITP_INVALID_VALUE    -999999

#define ITP_MAX_TRAFFIC      159
#define ITP_MAX_REFERENCE    2

/* Default parameter values */
#define ITP_FL_DEADBAND      1       /* 100ft */
#define ITP_VS_DEADBAND      300.0   /* ft/min */
```

```
#define ITP_MAX_RANGE        160.0  /* nmi */
#define ITP_MAX_LATERAL_OFFSET 15.0 /* nmi */

/* ITP initiation criteria constants */
#define ITP_RANGE_1          15.0   /* nmi   */
#define ITP_GS_DIFF_1        20.0   /* knots */
#define ITP_RANGE_2          20.0   /* nmi   */
#define ITP_GS_DIFF_2        30.0   /* knots */

/*-------------------------------------
  Enumerated types
  -------------------------------------*/
typedef enum {itpQualified, itpNonQualified} itpDataQuality;

typedef enum {itpNone,
              itpFollowingClimb, itpFollowingDescent,
              itpLeadingClimb, itpLeadingDescent,
              itpCombinedLeadingFollowingClimb,
              itpCombinedLeadingFollowingDescent
             } itpManeuver;

typedef enum {itpOwnBehindTraffic, itpOwnAheadTraffic,
              itpNoRelPos} itpRelPos;

typedef enum {itpSameDirection, itpOppositeDirection,
              itpCrossDirection} itpDirection;

/*-------------------------------------
   Type of traffic aircraft
  -------------------------------------*/
typedef unsigned int itpAcType;

/*-------------------------------------
  Substitute types
  -------------------------------------*/
typedef char  itpIdentifier[ITP_STR_LEN];
typedef int   itpAltitude; /* hundreds of feet */
typedef int   itpBool;

#define ITP_FALSE 0
#define ITP_TRUE 1

/*-------------------------------------
  Traffic/aircraft record
```

```
  ------------------------------------*/
typedef struct {
  itpIdentifier id;    /* flight identifier */
  double        lat;   /* latitude, in degrees: -90 ≤ lat ≤  90 */
  double        lon;   /* longitude, in degrees: -180 ≤ lon < 180 */
                       /* (west is negative)                      */
  itpAltitude   alt;   /* altitude, in 100s ft: alt > 0 */
  double        gs;    /* ground speed, in knots: gs > 0 */
  double        vs;    /* vertical speed, in ft/min */
  double        trk;   /* ground true track, in degrees 0 ≤ trk < 360 */
  itpDataQuality qlty; /* data quality */
} itpAircraft;

/*---------------------------------------
  Input lists
  ---------------------------------------*/
typedef struct {
  itpAircraft ac[ITP_MAX_TRAFFIC];
  int len;
} itpAircraftList;

/*---------------------------------------
  Record-type to bundle inputs together
  ---------------------------------------*/
typedef struct {
  itpAircraft    ownship;
  itpAircraftList traffic;
  itpAltitude    requested_fl;
} itpInputs;

/*---------------------------------------
  Output lists
  ---------------------------------------*/
typedef struct {
  itpIdentifier id;
  double       range;   /* ITP range to ownship */
  double       gsdiff;  /* Ground speed differential */
  double       offset;  /* Lateral offset */
  itpRelPos    relpos;  /* Traffic relative position to ownship */
  itpDirection dir;     /* Direction: same, opposite, cross */
  itpAcType    actype;  /* Type of aircraft. This is a bitmask that */
                        /* must be decoded through the functions: */
                        /* itpIsNearby,itpIsPotentiallyBlocking, */
                        /* itpIsNearest,itpIsReference          */
```

```
                        /* itpIsBlocking, itpIsManeuvering,     */
                        /* itpIsTooClose, itpIsTooFast          */
                        /* itpIsUnqualifiedData                 */
} itpTraffic;

typedef struct {
  itpTraffic trf[ITP_MAX_TRAFFIC];
  int len;
} itpTrafficList;

typedef int itpTrafficIdx; /* Index of aircraft */

typedef struct {
  itpTrafficIdx nrbyx[ITP_MAX_TRAFFIC];
  int len;
} itpNearbyList;

typedef struct {
  itpTrafficIdx blckx[ITP_MAX_TRAFFIC];
  int len;
} itpBlockingList;

typedef struct {
  itpTrafficIdx nrstx[ITP_MAX_REFERENCE];
  int len;
} itpNearestList;

typedef struct {
  itpTrafficIdx refx[ITP_MAX_REFERENCE];
  int len;
} itpReferenceList;

/*---------------------------------------
  Record-type to bundle outputs together
  ---------------------------------------*/
typedef struct {
  itpTrafficList   traffic;
  itpNearbyList    nearby;
  itpBlockingList  pblocking;
  itpNearestList   nearest;
  itpReferenceList reference;
  itpManeuver      maneuver;
} itpOutputs;
```

```
void itpEmptyOutputs(itpOutputs *outputs);

/*-----------------------------------------------
  Function to reset the global parameters
  -----------------------------------------------*/

void itpResetGlobalParameters();

/*------------------------------
  Get current parameter values
  ------------------------------*/

itpAltitude itpGetGlobalFlDeadband();
double      itpGetGlobalVsDeadband();
double      itpGetGlobalMaxRange();
double      itpGetGlobalMaxLateralOffset();
double      itpGetGlobalRange1();
double      itpGetGlobalGsDiff1();
double      itpGetGlobalRange2();
double      itpGetGlobalGsDiff2();

/*-----------------------------------------------
  Functions to set parameters to custom values
  -----------------------------------------------*/

void itpSetGlobalFlDeadband(itpAltitude);
void itpSetGlobalVsDeadband(double);
void itpSetGlobalMaxRange(double);
void itpSetGlobalMaxLateralOffset(double);
void itpSetGlobalRange1(double);
void itpSetGlobalGsDiff1(double);
void itpSetGlobalRange2(double);
void itpSetGlobalGsDiff2(double);

#endif
```

A.2 Global Parameters

```
/*=======================================================
  ITP algorithm global parameters
  Author: Cesar A. Munoz (NIA)
          Radu I. Siminiceanu (NIA)
```

```
  File   : ITP_Parameters.c
  Release: ITP_Module-6.d (10/11/06)
  ====================================================*/

#include "ITP_Defs.h"

struct {
  itpAltitude FlDeadband; /* flight level deadband (100s ft) */
  double VsDeadband;    /* vertical speed deadband (ft/min) */
  double MaxRange;      /* maximum range (nmi) */
  double MaxLateralOffset; /* maximum lateral offset for same vicinity (nmi) */
  /* ITP initiation criteria */
  double Range1;
  double GsDiff1;
  double Range2;
  double GsDiff2;
} itpGlobalParameters = {
  ITP_FL_DEADBAND,
  ITP_VS_DEADBAND,
  ITP_MAX_RANGE,
  ITP_MAX_LATERAL_OFFSET,
  ITP_RANGE_1,
  ITP_GS_DIFF_1,
  ITP_RANGE_2,
  ITP_GS_DIFF_2
  };

/*-----------------------------------------------
  function to reset the global parameters
  -----------------------------------------------*/

void itpResetGlobalParameters() {
  itpGlobalParameters.FlDeadband    = ITP_FL_DEADBAND;
  itpGlobalParameters.VsDeadband    = ITP_VS_DEADBAND;
  itpGlobalParameters.MaxRange      = ITP_MAX_RANGE;
  itpGlobalParameters.MaxLateralOffset = ITP_MAX_LATERAL_OFFSET;
  itpGlobalParameters.Range1        = ITP_RANGE_1;
  itpGlobalParameters.GsDiff1       = ITP_GS_DIFF_1;
  itpGlobalParameters.Range2        = ITP_RANGE_2;
  itpGlobalParameters.GsDiff2       = ITP_GS_DIFF_2;
}

/*-----------------------------------------------
  Functions to set global parameters
```

```
  ------------------------------------------------*/

void itpSetGlobalFlDeadband(itpAltitude fdb) {
  itpGlobalParameters.FlDeadband = fdb;
}

void itpSetGlobalVsDeadband(double vdb) {
  itpGlobalParameters.VsDeadband = vdb;
}

void itpSetGlobalMaxRange(double maxr) {
  itpGlobalParameters.MaxRange = maxr;
}

void itpSetGlobalMaxLateralOffset(double latoff) {
  itpGlobalParameters.MaxLateralOffset = latoff;
}

void itpSetGlobalRange1(double range1) {
  itpGlobalParameters.Range1 = range1;
}

void itpSetGlobalGsDiff1(double gs1) {
  itpGlobalParameters.GsDiff1 = gs1;
}

void itpSetGlobalRange2(double range2) {
  itpGlobalParameters.Range2 = range2;
}

void itpSetGlobalGsDiff2(double gs2) {
  itpGlobalParameters.GsDiff2 = gs2;
}

/*------------------------------
  Get current parameter values
  ------------------------------*/

itpAltitude itpGetGlobalFlDeadband() {
  return itpGlobalParameters.FlDeadband;
}

double itpGetGlobalVsDeadband() {
  return itpGlobalParameters.VsDeadband;
```

```
}

double itpGetGlobalMaxRange() {
  return itpGlobalParameters.MaxRange;
}

double itpGetGlobalMaxLateralOffset() {
  return itpGlobalParameters.MaxLateralOffset;
}

double itpGetGlobalRange1() {
  return itpGlobalParameters.Range1;
}

double itpGetGlobalGsDiff1() {
  return itpGlobalParameters.GsDiff1;
}

double itpGetGlobalRange2() {
  return itpGlobalParameters.Range2;
}

double itpGetGlobalGsDiff2() {
  return itpGlobalParameters.GsDiff2;
}
```

A.3 Program Interface

```
/*=========================================================
  ITP algorithm main file
  Author: Cesar A. Munoz (NIA)
          Radu I. Siminiceanu (NIA)

  File   : ITP_Algorithm.h
  Release: ITP_Module-6.d (10/11/06)
  =========================================================*/

#ifndef ITP_ALG
#define ITP_ALG

#include "ITP_Defs.h"

/* Is traffic closing with respect to ownship ?*/
itpBool itpIsClosing(itpTraffic trf);
```

```
/* Blocking type recognizers */
itpBool itpIsNearby(itpTraffic trf);
itpBool itpIsPotentiallyBlocking(itpTraffic trf);
itpBool itpIsNearest(itpTraffic trf);
itpBool itpIsReference(itpTraffic trf);
itpBool itpIsBlocking(itpTraffic trf);
itpBool itpIsManeuvering(itpTraffic trf);
itpBool itpIsUnqualifiedData(itpTraffic trf);
itpBool itpIsTooClose(itpTraffic trf);
itpBool itpIsTooFast(itpTraffic trf);

/* Decodes actype */
void itpBlckCode2String(itpIdentifier s,itpTraffic trf);

/* ITP main function */
void itpAlgorithm(itpInputs, itpOutputs*);

#endif
```

A.4 Code

```
/*=========================================================
  ITP algorithm main file
  Author: Cesar A. Munoz (NIA)
          Radu I. Siminiceanu (NIA)

  File   : ITP_Algorithm.c
  Release: ITP_Module-6.d (10/11/06)
  =========================================================*/

#include "ITP_Aviation.h"
#include "ITP_Algorithm.h"
#include <string.h>

/*=========================================================
  Internal constants: Type of traffic aircraft
  =========================================================*/

#define ITP_TRAFFIC            0x0
#define ITP_NEARBY             0x1
#define ITP_POTENTIALLY_BLOCKING 0x2
#define ITP_NEAREST            0x4
#define ITP_REFERENCE          0x8
```

```
#define ITP_BLOCKING           0x10
#define ITP_MANEUVERING        0x20
#define ITP_TOO_FAST           0x40
#define ITP_TOO_CLOSE          0x80
#define ITP_UNQUALIFIED_DATA   0x100

/*=========================================================
  Auxiliary functions
  =========================================================*/

double itpGeoRange(itpAircraft ac1,itpAircraft ac2) {
  double lt1, ln1, lt2, ln2;
  lt1 = ac1.lat*ITP_RAD_PER_DEG;
  ln1 = ac1.lon*ITP_RAD_PER_DEG;
  lt2 = ac2.lat*ITP_RAD_PER_DEG;
  ln2 = ac2.lon*ITP_RAD_PER_DEG;
  return ITP_ABS(itpDistance(lt1,ln1,lt2,ln2))*ITP_NMI_PER_RAD;
}

itpDirection itpGetDirection(double trk1, double trk2, double upto) {
  double delta;
  trk1 = itpTo360(trk1);
  trk2 = itpTo360(trk2);
  delta= ITP_MAX(trk1,trk2) - ITP_MIN(trk1,trk2);
  if (delta ≤ upto || delta ≥ 360-upto)
    return itpSameDirection;
  if (180-upto ≤ delta && delta ≤ 180+upto)
    return itpOppositeDirection;
  return itpCrossDirection;
}

void itpGetTrafficData(itpInputs inputs, itpOutputs *outputs, int i) {
  itpAircraft own,ac;
  double lta, lna, lto, lno, crsa, crso, lti, lni, trkoa;
  double dai,doi;
  int    eps;
  int    inter;

  own = inputs.ownship;
  ac  = inputs.traffic.ac[i];
  lto  = own.lat*ITP_RAD_PER_DEG;
  lno  = own.lon*ITP_RAD_PER_DEG;
  crso = own.trk*ITP_RAD_PER_DEG;
  lta  = ac.lat*ITP_RAD_PER_DEG;
```

```
  lna  = ac.lon*ITP_RAD_PER_DEG;
  crsa = ac.trk*ITP_RAD_PER_DEG;
  inter = itpIntersection(lta,lna,crsa,lto,lno,crso,&lti,&lni);
  /* printf("inter: %d (%6.2f,%6.2f)\n",inter,
     lti*ITP_DEG_PER_RAD,lni*ITP_DEG_PER_RAD); */
  if (inter < 0) {
    /* No intersection */
    trkoa = itpCourse(lto,lno,lta,lna)*ITP_DEG_PER_RAD;
    eps = (itpGetDirection(own.trk,trkoa,90) == itpSameDirection? 1: -1);
    outputs->traffic.trf[i].range =
      eps*ITP_ABS(itpDistance(lta,lna,lto,lno))*ITP_NMI_PER_RAD;
    outputs->traffic.trf[i].offset = 0;
  } else {
    doi = itpDistance(lto,lno,lti,lni);
    dai = itpDistance(lta,lna,lti,lni);
    outputs->traffic.trf[i].offset =
      itpCrossTrackError(lto,lno,lta,lna,lti,lni)*ITP_NMI_PER_RAD;
    /* Intersection behind of traffic */
    if (inter == 1 || inter == 3)
      dai *= -1;
    /* Intersection behind of ownship */
    if (inter == 2 || inter == 3)
      doi *= -1;
    if (outputs->traffic.trf[i].dir == itpSameDirection)
      outputs->traffic.trf[i].range = (doi - dai)*ITP_NMI_PER_RAD;
    else
      outputs->traffic.trf[i].range = (doi + dai)*ITP_NMI_PER_RAD;
  }
  outputs->traffic.trf[i].relpos = outputs->traffic.trf[i].range < 0 ?
    itpOwnAheadTraffic : itpOwnBehindTraffic;
  if (outputs->traffic.trf[i].dir == itpSameDirection)
    outputs->traffic.trf[i].gsdiff = ITP_SIGN(outputs->traffic.trf[i].range)*
      (inputs.ownship.gs - inputs.traffic.ac[i].gs);
  else
    outputs->traffic.trf[i].gsdiff = ITP_SIGN(outputs->traffic.trf[i].range)*
      (inputs.ownship.gs + inputs.traffic.ac[i].gs);
}

itpAltitude itpGetFlightLevel(itpAircraft ac) {
  int err = ac.alt % 10;
  int fl = ac.alt / 10;
  if (err ≤ itpGetGlobalFlDeadband())
    return 10*fl;
  if (err ≥ 10-itpGetGlobalFlDeadband())
```

```
    return 10*(fl+1);
  return ITP_INVALID_VALUE;
}

itpBool itpIsManeuveringAircraft(itpAircraft ac) {
  return
    itpGetFlightLevel(ac) == ITP_INVALID_VALUE ||
    ITP_ABS(ac.vs) > itpGetGlobalVsDeadband();
}

itpBool itpIsAtInterveningFlightLevels(itpInputs inputs, itpAircraft ac) {
  int eps;
  itpAltitude flown;

  flown = itpGetFlightLevel(inputs.ownship);
  eps = ITP_SIGN(inputs.requested_fl - flown);
  return
    flown != ITP_INVALID_VALUE &&
    eps*flown < eps*ac.alt &&
    eps*ac.alt < eps*inputs.requested_fl;
}

void itpGetNearby(itpInputs inputs,
                  itpOutputs *outputs) {
  int i;

  outputs->traffic.len = 0;
  outputs->nearby.len = 0;
  for (i=0; i < inputs.traffic.len; i++) {
    outputs->traffic.trf[i].actype = ITP_TRAFFIC;
    strcpy(outputs->traffic.trf[i].id,inputs.traffic.ac[i].id);
    outputs->traffic.trf[i].dir = itpGetDirection(inputs.traffic.ac[i].trk,
                                                  inputs.ownship.trk,45);
    if (outputs->traffic.trf[i].dir == itpSameDirection ||
        outputs->traffic.trf[i].dir == itpOppositeDirection) {
      itpGetTrafficData(inputs,outputs,i);
      /* if same vicinity */
      if (itpGeoRange(inputs.traffic.ac[i],inputs.ownship) <=
          itpGetGlobalMaxRange() &&
          outputs->traffic.trf[i].offset ≤ itpGetGlobalMaxLateralOffset()) {
        outputs->traffic.trf[i].actype |= ITP_NEARBY;
        outputs->nearby.nrbyx[outputs->nearby.len] = i;
        outputs->nearby.len++;
      }
```

```
    } else {
      outputs->traffic.trf[i].offset = ITP_INVALID_VALUE;
      outputs->traffic.trf[i].range = ITP_INVALID_VALUE;
      outputs->traffic.trf[i].gsdiff = ITP_INVALID_VALUE;
      outputs->traffic.trf[i].relpos = itpNoRelPos;
    }
    outputs->traffic.len++;
  }
}

itpBool itpIsClosing(itpTraffic trf) {
  return trf.gsdiff > 0;
}

itpAcType itpGetBlockingType(itpInputs inputs,
                             itpTrafficList traffic,
                             itpTrafficIdx idx) {
  itpAircraft ac;
  itpTraffic trf;
  itpAcType  actype;

  actype = 0x0;
  ac = inputs.traffic.ac[idx];
  trf = traffic.trf[idx];

  if (itpIsManeuveringAircraft(ac))
    actype |= ITP_MANEUVERING;
  if (ac.qlty != itpQualified)
    actype |= ITP_UNQUALIFIED_DATA;
  if (ITP_ABS(trf.range) < itpGetGlobalRange1())
    actype |= ITP_TOO_CLOSE;
  if (itpIsClosing(trf)) {
    if (trf.gsdiff > itpGetGlobalGsDiff2() ||
        ITP_ABS(trf.range) ≥ itpGetGlobalRange1() &&
        ITP_ABS(trf.range) < itpGetGlobalRange2() &&
        trf.gsdiff > itpGetGlobalGsDiff1())
      actype |= ITP_TOO_FAST;
    if (trf.gsdiff > itpGetGlobalGsDiff1() &&
        trf.gsdiff ≤ itpGetGlobalGsDiff2() &&
        ITP_ABS(trf.range) < itpGetGlobalRange2())
      actype |= ITP_TOO_CLOSE;
  }
  if (!actype)
    actype = ITP_BLOCKING;
```

```
  return ITP_POTENTIALLY_BLOCKING | actype ;
}

void itpBlckCode2String(itpIdentifier s,itpTraffic trf) {
  strcpy(s,"");
  if (itpIsBlocking(trf))
    strcat(s,"B");
  if (itpIsManeuvering(trf))
    strcat(s,"M");
  if (itpIsUnqualifiedData(trf))
    strcat(s,"U");
  if (itpIsTooClose(trf))
    strcat(s,"C");
  if (itpIsTooFast(trf))
    strcat(s,"F");
}

itpBool itpIsNearby(itpTraffic trf) {
  return trf.actype & ITP_NEARBY;
}

itpBool itpIsPotentiallyBlocking(itpTraffic trf) {
  return trf.actype & ITP_POTENTIALLY_BLOCKING;
}

itpBool itpIsNearest(itpTraffic trf) {
  return trf.actype & ITP_NEAREST;
}

itpBool itpIsReference(itpTraffic trf) {
  return trf.actype & ITP_REFERENCE;
}

itpBool itpIsBlocking(itpTraffic trf) {
  return trf.actype & ITP_BLOCKING;
}

itpBool itpIsManeuvering(itpTraffic trf) {
  return trf.actype & ITP_MANEUVERING;
}

itpBool itpIsTooClose(itpTraffic trf) {
  return trf.actype & ITP_TOO_CLOSE;
}
```

```
itpBool itpIsTooFast(itpTraffic trf) {
  return trf.actype & ITP_TOO_FAST;
}

itpBool itpIsUnqualifiedData(itpTraffic trf) {
  return trf.actype & ITP_UNQUALIFIED_DATA;
}

void itpGetPotentiallyBlocking(itpInputs inputs,
                               itpOutputs *outputs) {
  int i,idx;

  outputs->pblocking.len = 0;
  for (i=0; i < outputs->nearby.len; i++) {
    idx = outputs->nearby.nrbyx[i];
    if (outputs->traffic.trf[idx].dir == itpSameDirection &&
        itpIsAtInterveningFlightLevels(inputs,inputs.traffic.ac[idx])) {
      outputs->traffic.trf[idx].actype |=
        itpGetBlockingType(inputs,outputs->traffic,idx);
      outputs->pblocking.blckx[outputs->pblocking.len] = idx;
      outputs->pblocking.len++;
    }
  }
}

void itpGetNearest(itpInputs inputs,
                   itpOutputs *outputs) {

  int i,j,k,idx,jdx;
  outputs->nearest.len = 0;
  /* Find nearest ITP_MAX_REFERENCE aircraft */
  for (i=0; i < outputs->pblocking.len; i++) {
    idx = outputs->pblocking.blckx[i];
    for (j=0; j < outputs->nearest.len; j++) {
      jdx = outputs->nearest.nrstx[j];
      if (ITP_ABS(outputs->traffic.trf[idx].range) <
          ITP_ABS(outputs->traffic.trf[jdx].range))
        break;
    }
    if (j < ITP_MAX_REFERENCE) {
      for(k=ITP_MIN(outputs->nearest.len,ITP_MAX_REFERENCE-1); k > j; k--)
        outputs->nearest.nrstx[k] = outputs->nearest.nrstx[k-1];
      outputs->nearest.nrstx[j]=idx;
```

```
      if (outputs->nearest.len < ITP_MAX_REFERENCE)
        outputs->nearest.len++;
    }
  }
  /* Remove nearest at the same fligh level same direction */
  idx=outputs->nearest.nrstx[0];
  for (i=0,j=1,idx=outputs->nearest.nrstx[0]; j < outputs->nearest.len; j++) {
    jdx = outputs->nearest.nrstx[j];
    if (itpGetFlightLevel(inputs.traffic.ac[idx]) !=
        itpGetFlightLevel(inputs.traffic.ac[jdx]) ||
        ITP_SIGN(outputs->traffic.trf[idx].range) !=
        ITP_SIGN(outputs->traffic.trf[jdx].range)) {
      outputs->traffic.trf[idx].actype |= ITP_NEAREST;
      idx = jdx;
      i++;
    }
  }
  if (i < outputs->nearest.len) {
    outputs->traffic.trf[idx].actype |= ITP_NEAREST;
    i++;
  }
  outputs->nearest.len = i;
}

void itpGetReference(itpInputs inputs,
                     itpOutputs *outputs) {

  int i,idx;
  itpGetNearest(inputs,outputs);
  outputs->reference.len = 0;
  for (i=0; i < outputs->nearest.len; i++) {
    idx = outputs->nearest.nrstx[i];
    if (!itpIsBlocking(outputs->traffic.trf[idx])) {
      outputs->reference.len = 0;
      break;
    } else {
      outputs->traffic.trf[idx].actype |= ITP_REFERENCE;
      outputs->reference.refx[outputs->reference.len] = idx;
      outputs->reference.len++;
    }
  }
}

void itpGetManeuver(itpInputs inputs,
```

```
                    itpOutputs *outputs) {
  itpBool climb,lead,follow;
  int i;

  outputs->maneuver = itpNone;
  lead = 0;
  follow = 0;
  climb = (inputs.requested_fl > inputs.ownship.alt);
  for (i=0; i<outputs->reference.len; i++) {
    if (outputs->traffic.trf[outputs->reference.refx[i]].relpos ==
        itpOwnAheadTraffic)
      lead = 1;
    else
      follow = 1;
  }
  if (lead && follow)
    outputs->maneuver = (climb? itpCombinedLeadingFollowingClimb :
                         itpCombinedLeadingFollowingDescent);
  else if (lead)
    outputs->maneuver = (climb? itpLeadingClimb : itpLeadingDescent);
  else
    outputs->maneuver = (climb? itpFollowingClimb : itpFollowingDescent);
}

/*-------------------------------------------------
  ITP main function.
  CAVEAT: itpGLobalParameters should be set before
          calling itpAlgorithm
  -------------------------------------------------*/

void itpSetOutputs(itpOutputs *outputs) {
  outputs->traffic.len = 0;
  outputs->nearby.len = 0;
  outputs->pblocking.len = 0;
  outputs->nearest.len = 0;
  outputs->reference.len = 0;
  outputs->maneuver = itpNone;
}

void itpAlgorithm(itpInputs inputs, itpOutputs *outputs) {
  itpSetOutputs(outputs);
  itpGetNearby(inputs,outputs);
  if (!itpIsManeuveringAircraft(inputs.ownship) &&
      outputs->nearby.len > 0) {
```

```
    itpGetPotentiallyBlocking(inputs,outputs);
    if (outputs->pblocking.len > 0) {
      itpGetReference(inputs,outputs);
      if (outputs->reference.len > 0)
        itpGetManeuver(inputs,outputs);
    }
  }
}
```

B AVIATION FORMULARY

Except for the function `ITP_intersection`, which was developed by R. Siminiceanu, all the other functions were taken from William's Aviation Formulary [3].

B.1 Program Interface

```
/*=========================================================
  Aviation Formulary
  Author: Cesar A. Munoz (NIA)
          Radu I. Siminiceanu (NIA)

  File   : ITP_Aviation.h
  Release: ITP_Module-6.d (10/11/06)
  =========================================================*/

#ifndef ITP_AVI
#define ITP_AVI

#include <math.h>

#define ITP_MAX(a,b) ((a) > (b) ? (a) : (b))
#define ITP_MIN(a,b) ((a) < (b) ? (a) : (b))
#define ITP_ABS(x) ((x) < 0 ? -(x) : x)
#define ITP_SQ(x)  (x*x)
#define ITP_SIGN(x) ((x) < 0 ? -1 : 1)
#define ITP_PI           3.14159265358979323846
#define ITP_RAD_PER_DEG  (ITP_PI/180.0)
#define ITP_DEG_PER_RAD  (180.0/ITP_PI)
#define ITP_FT_PER_NMI   6076.11548556431
#define ITP_EARTH_RADIUS_FT 20898908.0
#define ITP_NMI_PER_RAD (ITP_EARTH_RADIUS_FT/ITP_FT_PER_NMI)
#define ITP_ALMOST_ZERO  0.00001

int    itpIsAlmostZero(double x);
double itpSqrtSafe(double x);
```

```
double itpAsinSafe(double x);
double itpAcosSafe(double x);
double itpAtanyxSafe(double y, double x);
double itpTo2pi(double x);
double itpTo360(double x);
double itpDistance(double lat1, double lon1, double lat2, double lon2);
double itpCourse(double lat1, double lon1, double lat2, double lon2);
int    itpIntersection(double lat1,double lon1,double crs13,
                       double lat2,double lon2,double crs23,
                       double *lati, double *loni);
double itpCrossTrackError(double lat1, double lon1,
                          double lat2, double lon2,
                          double lat3, double lon3);
#endif
```

B.2 Code

```
/*============================================================
  These formulas are taken from Aviation Formulary v1.42:
  http://williams.best.vwh.net
  - all units are in radians
  - modified to convention "east is positive"

  Author: Cesar A. Munoz (NIA)
          Radu I. Siminiceanu (NIA)

  File   : ITP_Aviation.c
  Release: ITP_Module-6.d (10/11/06)
  ============================================================*/

#include <stdio.h>
#include <stdlib.h>

#include "ITP_Aviation.h"

int itpIsAlmostZero(double x) {
  return ITP_ABS(x) < ITP_ALMOST_ZERO;
}

double itpSqrtSafe(double x) {
  return sqrt(ITP_MAX(x,0.0));
}

double itpAsinSafe(double x) {
```

```
  return asin(ITP_MAX(-1.0,ITP_MIN(x,1.0)));
}

double itpAcosSafe(double x) {
  return acos(ITP_MAX(-1.0,ITP_MIN(x,1.0)));
}

double itpAtanyxSafe(double y, double x) {
  if (itpIsAlmostZero(x) && itpIsAlmostZero(y))
    return 0.0;
  return atan2(y,x);
}

double itpTo2pi(double x) {
  double mod = x - 2*ITP_PI*((int) (x/(2*ITP_PI)));
  if (mod < 0) mod += 2*ITP_PI;
  return mod;
}

double itpTo360(double x) {
  double mod = x - 360.0*((int) (x/360.0));
  if (mod < 0) mod += 360.0;
  return mod;
}

double itpDistance(double lat1, double lon1, double lat2, double lon2) {
  return 2*itpAsinSafe(itpSqrtSafe(ITP_SQ(sin((lat1-lat2)/2)) +
                                   cos(lat1)*cos(lat2)*sin((lon2-lon1)/2)*
                                   sin((lon2-lon1)/2)));
}

double itpCourse(double lat1, double lon1, double lat2, double lon2) {
  if (itpIsAlmostZero(cos(lat1)))
    return (lat1>0 ? ITP_PI : 0.0);
  return itpTo2pi(itpAtanyxSafe(sin(lon2-lon1)*cos(lat2),
                                cos(lat1)*sin(lat2)-
                                sin(lat1)*cos(lat2)*cos(lon2-lon1)));
}

/* This function returns:
   -1 : no intersection,
   0 : ahead of both,
   1 : behind of 1, ahead of 2
   2 : behind of 2, ahead of 1
```

```
   3 : behind of both
   If positive (*ilat,*ilon) is the intersection point */
int itpIntersection(double lat1, double lon1, double crs13,
                    double lat2, double lon2, double crs23,
                    double *ilat, double *ilon) {
  double crs12, crs21, ang1, ang2, ang3, dst12, dst13, lat3, lon3;
  double modcrs13, modcrs23, delta;
  int flipped1, flipped2;

  crs13 = itpTo2pi(crs13);
  crs23 = itpTo2pi(crs23);
  dst12 = itpDistance(lat1, lon1, lat2, lon2);
  crs12 = itpCourse(lat1, lon1, lat2, lon2);
  crs21 = itpCourse(lat2, lon2, lat1, lon1);

  ang1=crs13-crs12;
  ang2=crs21-crs23;

  if (ITP_ABS(sin(ang1)) < 0.1 &&
      ITP_ABS(sin(ang2)) < 0.1)
    /*  if (itpIsAlmostZero(sin(ang1)) && itpIsAlmostZero(sin(ang2))) */
    return -1;

  flipped1 = 0;
  flipped2 = 0;
  modcrs13 = crs13;
  modcrs23 = crs23;

  if (0<=crs13 && crs13<=ITP_PI) { /* ownship going east */
    if (crs13 ≤ crs21 && crs21 ≤ crs13+ITP_PI) { /* target on left side */
      if (crs23 > crs13+ITP_PI || crs23 < crs13) {
        modcrs23 = itpTo2pi(crs23 + ITP_PI);
        flipped2 = 1;
      }
      flipped1 = (crs21 < modcrs23 && modcrs23 < crs13+ITP_PI);
    }
    else { /* target on the right side */
      if (crs23 > crs13 && crs23 < crs13+ITP_PI) {
        modcrs23 = itpTo2pi(crs23 + ITP_PI);
        flipped2 = 1;
      }
      if (itpIsAlmostZero(ITP_ABS(crs21))) crs21 = crs21 + 2*ITP_PI;
      flipped1 = ((crs21 > crs13+ITP_PI &&
                   crs13+ITP_PI < modcrs23 && modcrs23 < crs21) ||
```

```
                (crs21 < crs13 && (modcrs23 > crs13+ITP_PI ||
                                   modcrs23 < crs21)));
  }
}
else { /* ownship going west */
  if (crs21 ≥ crs13 || crs21 ≤ crs13-ITP_PI) {/* target on the left side */
    if (crs13-ITP_PI < crs23 && crs23 < crs13) {
      modcrs23 = itpTo2pi(crs23 + ITP_PI);
      flipped2 = 1;
    }
    flipped1 = ((crs21 > crs13 && (modcrs23 > crs21 ||
                                   modcrs23 < crs13-ITP_PI)) ||
                (crs21 < crs13-ITP_PI && modcrs23 > crs21 &&
                 modcrs23 < crs13-ITP_PI));
  }
  else { /* target on the right side */
    if (crs23 > crs13 || crs23  < crs13-ITP_PI) {
      modcrs23 = itpTo2pi(crs23 + ITP_PI);
      flipped2 = 1;
    }
    if (itpIsAlmostZero(ITP_ABS(crs21))) crs21 = crs21 + 2*ITP_PI;
    flipped1 = (modcrs23 > crs13-ITP_PI && modcrs23 < crs21);
  }
}

ang1  = itpTo2pi(crs13-crs12+ITP_PI) - ITP_PI;
ang2  = itpTo2pi(crs21-modcrs23+ITP_PI) - ITP_PI;
ang1  = ITP_ABS(ang1);
ang2  = ITP_ABS(ang2);

if (flipped1) {
  ang1    = itpTo2pi(ITP_PI - ang1);
  modcrs13 = itpTo2pi(crs13 + ITP_PI);
}

ang3 = itpAcosSafe(-cos(ang1)*cos(ang2) + sin(ang1)*sin(ang2)*cos(dst12));
if (itpIsAlmostZero(sin(ang3))) {
  *ilat = 0.0;
  *ilon = 0.0;
  return -1;
}

dst13 = itpAsinSafe(sin(ITP_ABS(ang2))*sin(dst12)/sin(ang3));
lat3  = itpAsinSafe(sin(lat1)*cos(dst13)+cos(lat1)*sin(dst13)*cos(modcrs13));
```

```
  delta = itpAsinSafe(sin(crs13)*sin(dst13)/cos(lat3));

  if (flipped1==0)
    lon3 = itpTo2pi(lon1 + delta + ITP_PI) - ITP_PI;
  else
    lon3 = itpTo2pi(lon1 - delta + ITP_PI) - ITP_PI;

  *ilat = lat3;
  *ilon = lon3;
  return 2*flipped2 + flipped1;
}

/* Track error to common point (lat3,lon3), wrt (lat2,lon2) */
double itpCrossTrackError(double lat1, double lon1,
                          double lat2, double lon2,
                          double lat3, double lon3) {
  double crs31, crs32,dst32;
  dst32 = itpDistance(lat3, lon3, lat2, lon2);
  if (itpIsAlmostZero(cos(lat3))) {
    /* (lat3,lon3) is a Pole */
    crs31 = lon1;
    crs32 = lon2;
  } else {
    crs31 = itpCourse(lat3, lon3, lat1, lon1);
    crs32 = itpCourse(lat3, lon3, lat2, lon2);
  }
  return ITP_ABS(itpAsinSafe(sin(dst32)*sin(crs32-crs31)));
}
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01/08/2007	Contractor Report	

4. TITLE AND SUBTITLE

In-Trail Procedure (ITP) Algorithm Design

5a. CONTRACT NUMBER

5b. GRANT NUMBER
NCC1-02043

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Cesar A. Munoz
Radu I. Siminiceanu

5d. PROJECT NUMBER
2204

5e. TASK NUMBER

5f. WORK UNIT NUMBER
411931.02.07.07

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

National Institute of Aerospace (NIA)
100 Exploration Way
Hampton, VA 23666

8. PERFORMING ORGANIZATION REPORT NUMBER
NIA Report No. 2007-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITOR'S ACRONYM(S)
NASA

11. SPONSORING/MONITORING REPORT NUMBER
NASA/CR-2007-214863

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 61
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

Langley Technical Monitor: Ricky Butler

14. ABSTRACT

The primary objective of this document is to provide a detailed description of the In-Trail Procedure (ITP) algorithm, which is part of the Airborne Traffic Situational Awareness – In-Trail Procedure (ATSA-ITP) application. To this end, the document presents a high level description of the ITP Algorithm and a prototype implementation of this algorithm in the programming language C.

15. SUBJECT TERMS

In-Trail Procedure; Algorithm Design

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19b. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
U	U	U	UU	38	19b. TELEPHONE NUMBER *(Include area code)* (301) 621-0390