AER E 407 Final Project Report

Meaghan McCleary

November 24, 2020

1 Abstract

This project seeks to verify a piece of software used to create customized flight plans for research aircraft executing mission to gather imagery of spacecraft targets in flight. The project takes a model-checking approach to the problem using the Spin model checker with integrated C functions for mathematical tasks. The specifications deal with how the user can input information into the tool and whether certain inputs can cause the flight path to represent physically impossible maneuvers for the aircraft. The results of the project are valuable in that they inform possible improvements to the tool that may make the process less prone to accidental ill-effects.

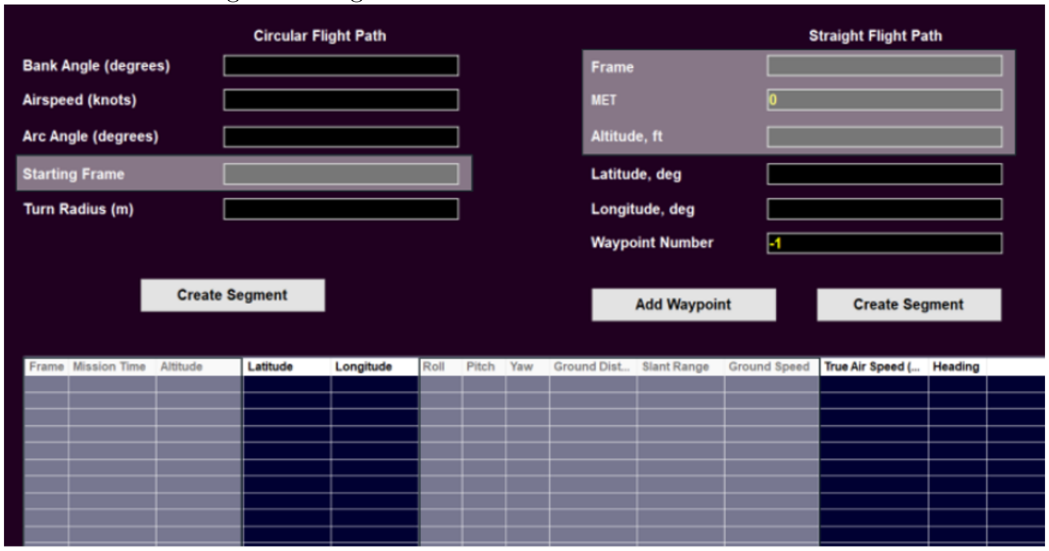
2 Motivation

The flight path planning software is part of a larger suite of tools used for mission planning. This comparatively small module and its verification serve as a proof of concept for verification of other portions of the software. These efforts will generate higher levels of confidence in the tools as the group grows its list of projects to include many different imagery targets and interested parties in both the government and private sectors.

3 System overview

The graphic below illustrates the appearance and functionality of the user interface of the tool. The grayed-out areas are the portions of the functionality I have chosen not to model because of their relevancy or for the sake of limiting the scope of the project in the interest of time.

Figure 1: Flight Path Planner User Interface



MET	0
Waypoint Number	-1

The interface works by allowing the user to alternate straight and curved flight path segments, such that the whole of the flight path is customizable to the mission being developed. The circular flight path generates a series of coordinates along a circle. The geometry of this circle is calculated form the bank angle and airspeed of the aircraft, and the arc angle determines the portion of the circle used in the path. The straight flight path is created between two sets of coordinates specified by the user as waypoints.

I have listed here the aspects of the tool included in the interface, their status related to the model, and why they have or have not been included.

Figure 2: Table of User Interface Elements

Item	Origin in Software	Model Inclusion	Rationale
Bank Angle	User Provided	True	Necessary for coordinate calculation
Airspeed	User Provided	True	Necessary for coordinate calculation
Arc Angle	User Provided	True	Necessary for coordinate calculation
Starting Frame	User Provided	False	Frame indexing causes state space explosion
Turn Radius	Calculated	Only in C	Intermediate calculation step
Frame	User Provided	False	Frame indexing causes state space explosion
MET (Mission Time)	Provided externally	False	Unnecessary information
Altitude	User Provided	False	Assumed constant
Latitude	User Provided	True	Necessary for specification
Longitude	User Provided	True	Necessary for specification
Waypoint Number	User Provided	False	Frame indexing causes state space explosion
Table of frame information	Automatically generated	False	Frame indexing causes state space explosion

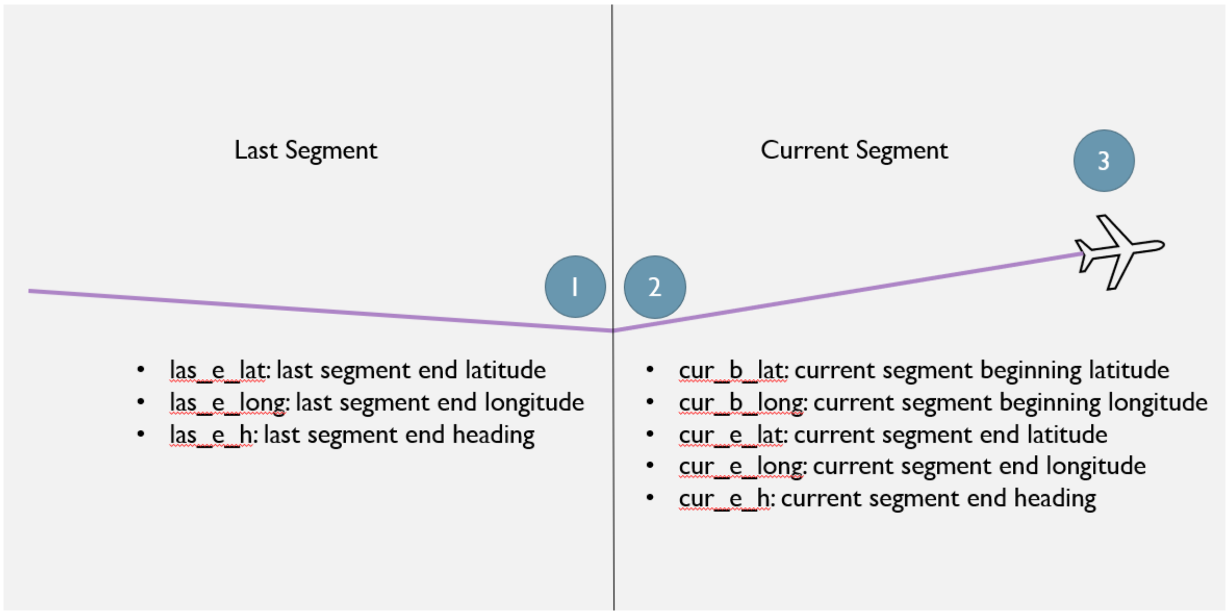
4 Model Development Process

I chose to write the model for this project in Promela and integrate the mathematical processes as C functions. The crux of the modelling problem here seemed to be creating processes that would accurately represent all the possible results from the possible user inputs. However, given the level of customizability possible with the tool, there were a few pitfalls possible that would have led to state space explosions. I will discuss them here.

4.1 Scope and State Space Limitation

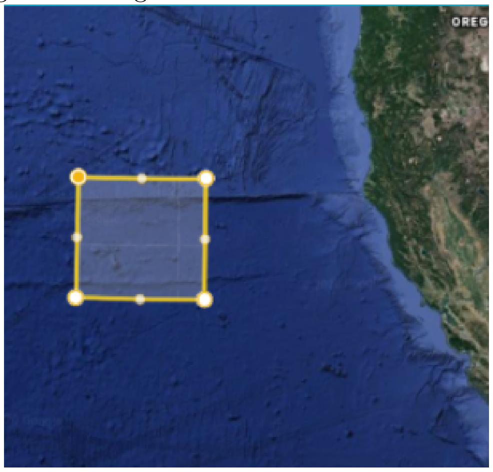
First, I needed to find a way to minimize how much information needed to be stored to describe the aircraft's position. In the tool itself, the goal is the creation of a complete trajectory describing the flight path, with each entry in the trajectory being a one second time step with accompanying information about the state of the aircraft at that particular time. This is obviously not a viable option for Spin/Promela, because the state space explosion would cause the model to be unrunnable. However, since my specifications focus on the relationships and boundaries between consecutive segments, I focused on storing only the information needed at the beginnings and ends of the segments and overwriting that information for each new segment, greatly reducing the variable list. I have included an illustration of this concept in the figure below.

Figure 3: Segment Variable Illustration


Last Segment
Current Segment
1
2
3
las_e_lat: last segment end latitude
las_e_long: last segment end longitude
las_e_h: last segment end heading
cur_b_lat: current segment beginning latitude
cur_b_long: current segment beginning longitude
cur_e_lat: current segment end latitude
cur_e_long: current segment end longitude
cur_e_h: current segment end heading

The second potential cause of a state space explosion was the fact that this tool could be used to plan a mission anywhere in the world. Unlimited location and coordinate resolution would lead to a state space that was effectively infinite, which would cause an obvious problem. To fix this, I allocated a 150 nautical mile by 150 nautical mile area off the west coast of the United States, pictured below, and also limited the resolution of the latitude and longitude values to one tenth of a degree.

Figure 4: Google Earth View of Allocated Area



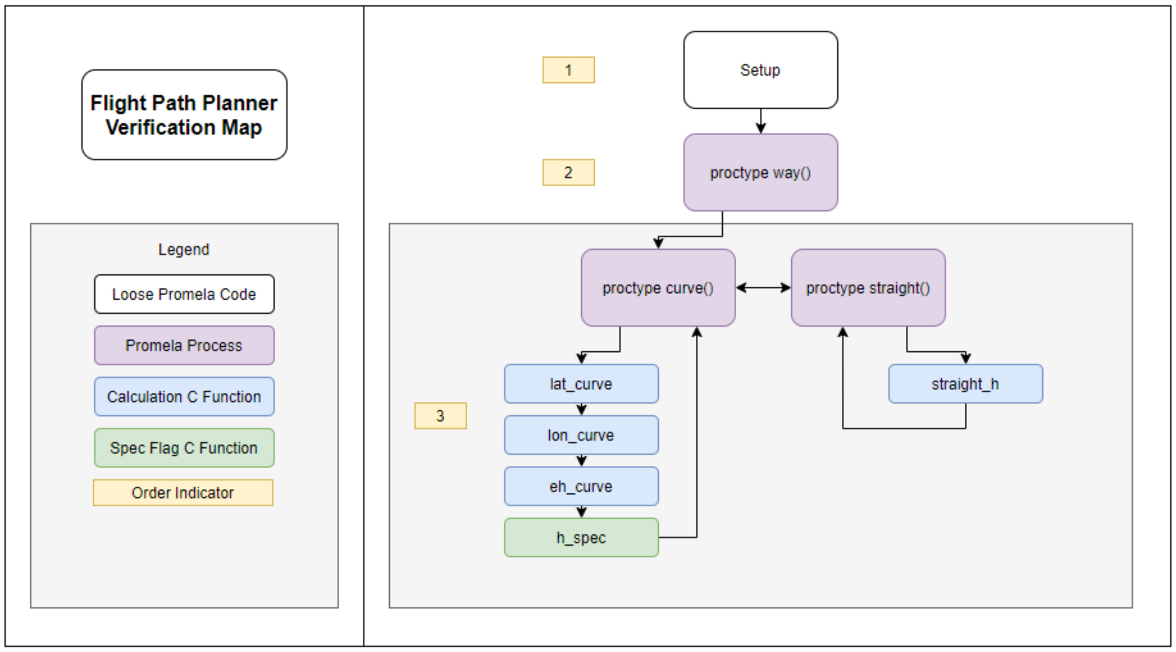
Third, for the purposes of proving the concept, I have chosen to focus on two specific pieces of information (location and heading) instead of attempting verification activities with the rest of the information that might typically be stored with the flight path.

Fourth and finally, I recognized the fact that the group uses several types of aircraft in their mission plans, and so I decided to select one of these with its corresponding characteristics in order to establish an acceptable range of user inputs. Inherent in this choice was also an assumption that the user, being an experienced member of the team, would understand that it was necessary to not specify that the aircraft, for example, travel faster than it's maximum speed or turn at a steeper bank angle than is reasonable. The aircraft I selected is the WB-57, a research aircraft based out of Ellington Field in Houston, Texas (WB-57, 2017).

4.2 Model Layout

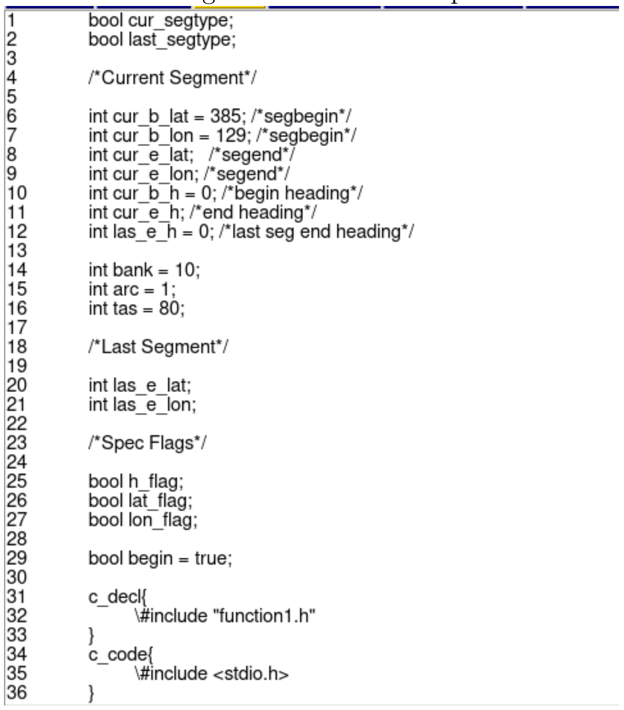
Below, I have included a concept map of my model, its various Promela processes and C-functions, and the relationships between them, and I have explained them in further detail in the following sections.

Figure 5: Model Map

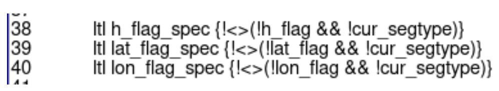


4.2.1 Setup

Figure 6: Promela Setup Section



```
bool cur_segtype;
bool last_segtype;

/*Current Segment*/

int cur_b_lat = 385; /*segbegin*/
int cur_b_lon = 129; /*segbegin*/
int cur_e_lat;   /*segend*/
int cur_e_lon; /*segend*/
int cur_b_h = 0; /*begin heading*/
int cur_e_h; /*end heading*/
int las_e_h = 0; /*last seg end heading*/

int bank = 10;
int arc = 1;
int tas = 80;

/*Last Segment*/

int las_e_lat;
int las_e_lon;

/*Spec Flags*/

bool h_flag;
bool lat_flag;
bool lon_flag;

bool begin = true;

c_decl{
        \#include "function1.h"
}
c_code{
        \#include <stdio.h>
}
```

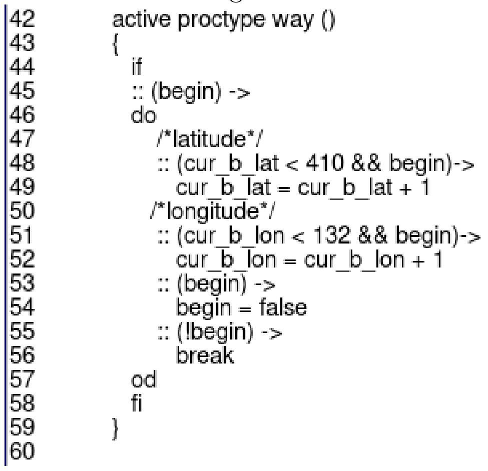


```
ltl h_flag_spec {!<>(!h_flag && !cur_segtype)}
ltl lat_flag_spec {!<>(!lat_flag && !cur_segtype)}
ltl lon_flag_spec {!<>(!lon_flag && !cur_segtype)}
```

This section accomplishes the initialization of all of the global variables and the integration of the relevant C libraries (My function1.h file and the standard stdio.h).

4.2.2 Proctype Way

Figure 7: Promela Proctype Way

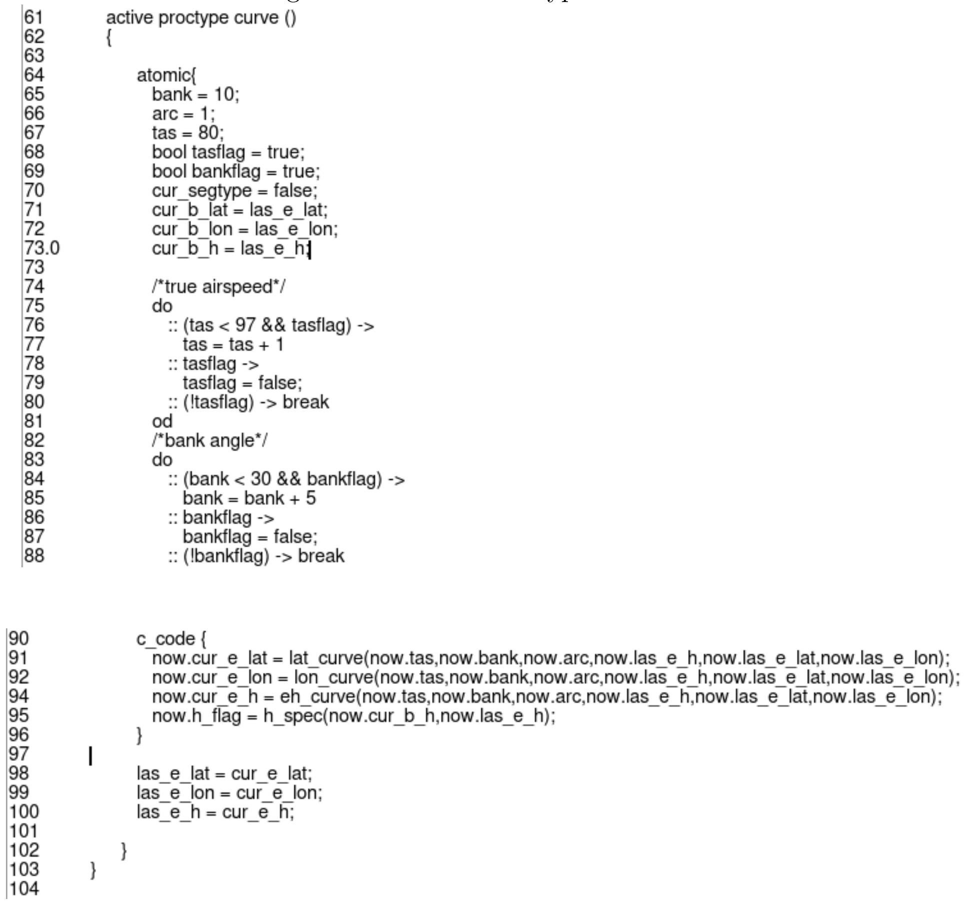


```
active proctype way ()
{
  if
  :: (begin) ->
  do
     /*latitude*/
     :: (cur_b_lat < 410 && begin)->
        cur_b_lat = cur_b_lat + 1
    /*longitude*/
     :: (cur_b_lon < 132 && begin)->
        cur_b_lon = cur_b_lon + 1
     :: (begin) ->
        begin = false
     :: (!begin) ->
        break
  od
  fi
}
```

This process is responsible for creating the first set of coordinates, which will be used in cases where the first segment is curved. The straight segment process has its own loop responsible for this. Note that the 410 and 132 values are the upper bounds of the latitude and longitude contained in the aforementioned square of ocean that represents the state space. This briefly presented a problem because I was not aware of Spin's inability to handle decimals, but I solved the problem by dividing and multiplying by 10 as values are passed in and out of the C functions.

4.2.3 Proctype Curve

Figure 8: Promela Proctype Curve



```
active proctype curve ()
{

    atomic{
      bank = 10;
      arc = 1;
      tas = 80;
      bool tasflag = true;
      bool bankflag = true;
      cur_segtype = false;
      cur_b_lat = las_e_lat;
      cur_b_lon = las_e_lon;
      cur_b_h = las_e_h;

      /*true airspeed*/
      do
        :: (tas < 97 && tasflag) ->
          tas = tas + 1
        :: tasflag ->
          tasflag = false;
        :: (!tasflag) -> break
      od
      /*bank angle*/
      do
        :: (bank < 30 && bankflag) ->
          bank = bank + 5
        :: bankflag ->
          bankflag = false;
        :: (!bankflag) -> break
```

```
    c_code {
      now.cur_e_lat = lat_curve(now.tas,now.bank,now.arc,now.las_e_h,now.las_e_lat,now.las_e_lon);
      now.cur_e_lon = lon_curve(now.tas,now.bank,now.arc,now.las_e_h,now.las_e_lat,now.las_e_lon);
      now.cur_e_h = eh_curve(now.tas,now.bank,now.arc,now.las_e_h,now.las_e_lat,now.las_e_lon);
      now.h_flag = h_spec(now.cur_b_h,now.las_e_h);
    }

    las_e_lat = cur_e_lat;
    las_e_lon = cur_e_lon;
    las_e_h = cur_e_h;

  }
}

```

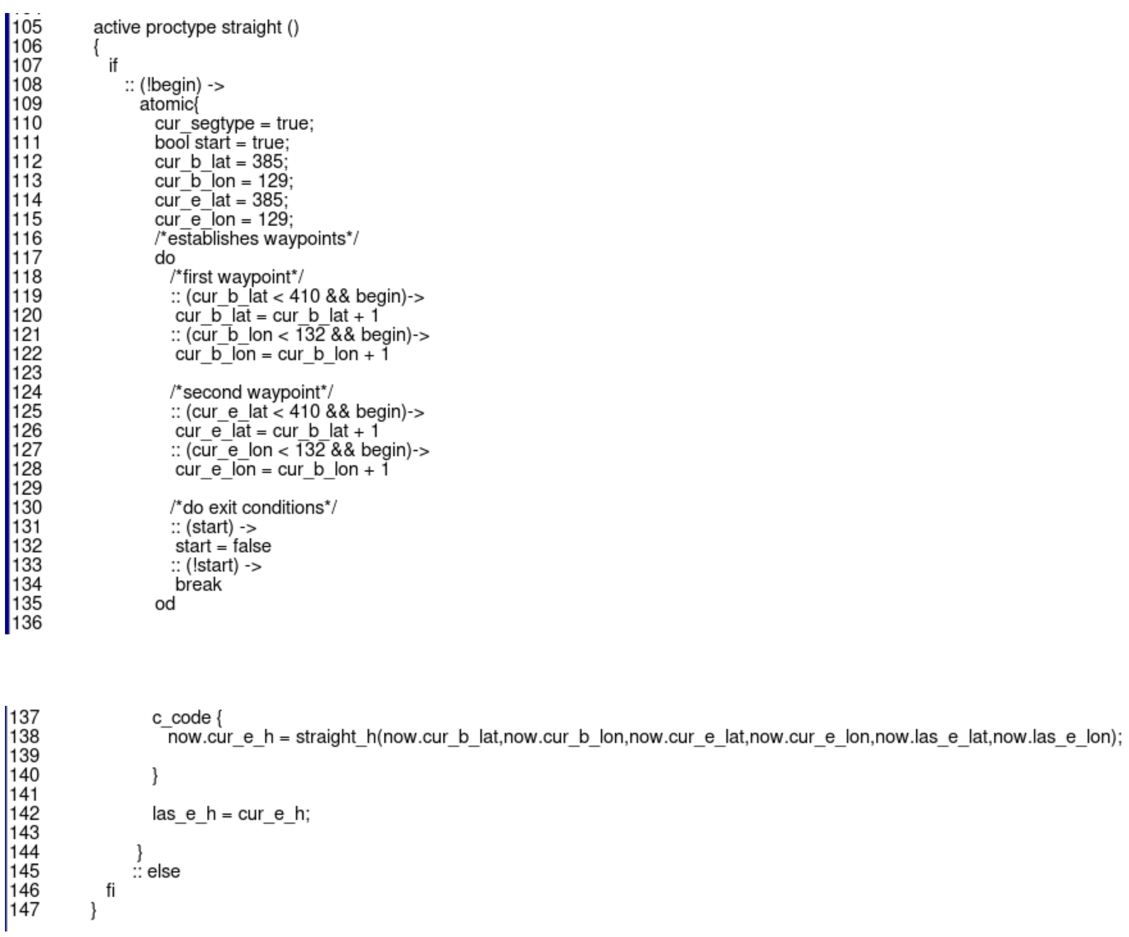
This process is responsible for creation of the curved segments. The first section is initialization of local variables and and value setting of global ones. The two do loops set values for the airspeed and bank angle characteristics of the segment. Their upper bounds are 97 meters per second and thirty degrees, respectively, which come from the WB-57 Experimenters Handbook (WB-57, 2017).

After the do-loops comes a series of C function calls. The first two (lat_curve and lon_curve) are responsible for calculating the final set of coordinates belonging to the current segment. The eh_curve function calculates the final heading value for use in the specification, and the h_spec function is responsible for assigning a value to the boolean flag used in the heading specification. I have elected not to include the C code here in the interest of space, but the file function1.h in the project repository includes all of the functions with comments.

Finally, the last three lines of the process transfer values from the current segment frame to the last segment frame.

4.2.4 Proctype Straight

Figure 9: Promela Proctype Straight



```
active proctype straight ()
{
  if
    :: (!begin) ->
      atomic{
        cur_segtype = true;
        bool start = true;
        cur_b_lat = 385;
        cur_b_lon = 129;
        cur_e_lat = 385;
        cur_e_lon = 129;
        /*establishes waypoints*/
        do
          /*first waypoint*/
          :: (cur_b_lat < 410 && begin)->
           cur_b_lat = cur_b_lat + 1
          :: (cur_b_lon < 132 && begin)->
           cur_b_lon = cur_b_lon + 1

          /*second waypoint*/
          :: (cur_e_lat < 410 && begin)->
           cur_e_lat = cur_b_lat + 1
          :: (cur_e_lon < 132 && begin)->
           cur_e_lon = cur_b_lon + 1

          /*do exit conditions*/
          :: (start) ->
           start = false
          :: (!start) ->
           break
        od



        c_code {
          now.cur_e_h = straight_h(now.cur_b_lat,now.cur_b_lon,now.cur_e_lat,now.cur_e_lon,now.las_e_lat,now.las_e_lon);

        }

        las_e_h = cur_e_h;

      }
     :: else
  fi
}
```

This process creates the straight fligh path segments. As before, the first section is variable assignments, followed by a do loop that creates a set of coordinates for the beginning of the segment and a segment for the end, mimicking the user input. This is followed by calling the function that calculates the heading at the end of the segment and the passing of this value to the last segment frame.

4.3 Specification Development

I went through multiple design iterations of specifications by the time this project was done. It was a challenge to create specifications that were relevant to the system functionality and verifiable. I will explain these iterations and the reasons why the design had to change in the following sections.

4.3.1 Iteration 1

I began with the premise that I wished to verify that the user could not create a flight path incompatible with physics or with the capabilities of the aircraft. To this end, I established a few initial specifications, which I have listed here along with the reasoning behind later changes.

- $\neg \diamond (Change\_In\_Heading > 15)$
- $\neg \diamond (Current\_Beginning\_Latitude! = Last\_End\_Latitude)$
- $\neg \diamond (Current\_Beginning\_Longitude! = Last\_End\_Longitude)$
- $\neg\diamond((True\_Air\_Speed > Max\_Air\_Speed) \vee (True\_Air\_Speed < Min\_Air\_Speed))$
- $\neg\diamond((Bank\_Angle > Max\_Bank\_Angle) \vee (Bank\_Angle < Min\_Bank\_Angle))$

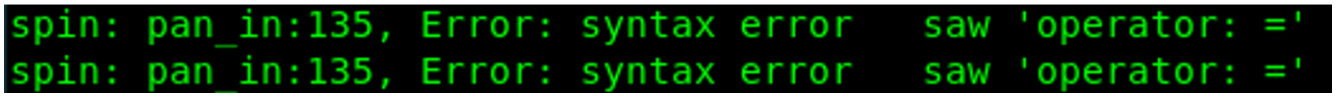
The problem that inspired the first round of changes was the realization that user input to the system is arbitrary and that it makes no sense to test air speed of bank angle because the system has no safeguards against unreasonable values. This led to changes in iteration 2.

4.3.2 Iteration 2

- $\neg \diamond (Change\_In\_Heading > 15)$
- $\neg \diamond (Current\_Beginning\_Latitude! = Last\_End\_Latitude)$
- $\neg \diamond (Current\_Beginning\_Longitude! = Last\_End\_Longitude)$

This iteration solves the problem described in iteration 1, but still doesn't function correctly. Using any of these specifications generates an error (pictured below) because LTL formulas cannot contain mathematical operators. The heading change specification also presents a problem because of the arbitrary nature of the straight segment creation process. It is obvious based on the design of the software that this specification will produce any number of counterexamples. Finally, it is not useful to apply the location specifications to the straight flight path segments because of the arbitrary user input issue.

Figure 10: LTL Math Error



```
spin: pan_in:135, Error: syntax error   saw 'operator: ='
spin: pan_in:135, Error: syntax error   saw 'operator: ='
```

Solving these issues leads to iteration 3.

4.3.3 Iteration 3

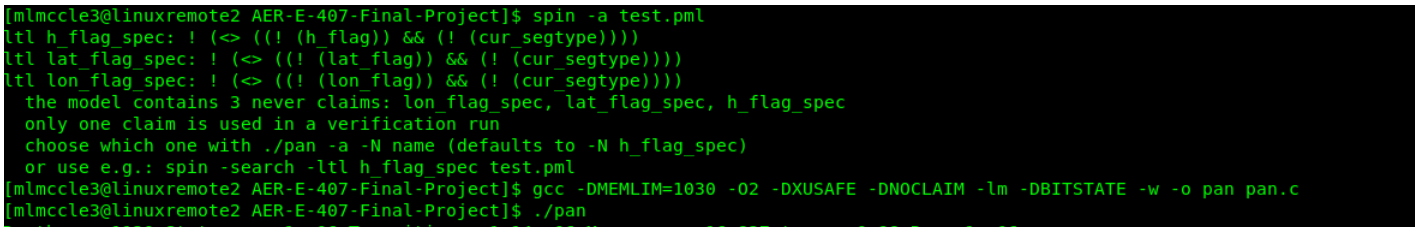
- $\neg \diamond (\neg Heading\_Flag \wedge \neg Cur\_SegType)$
- $\neg \diamond (\neg Latitude\_Flag \wedge \neg Cur\_SegType)$
- $\neg \diamond (\neg Longitude\_Flag \wedge \neg Cur\_SegType)$

In this iteration, the inequalities in the previous iteration have been replaced by boolean flags. These flags are generated via C functions that represent the same concept. Also, the specification have been changed to apply only to curved segments. Finally, the heading specification has been altered such that it specifies that the beginning heading of a curved segment should match the ending heading of the previous segment.

5 Verification and Results

Below is an image of the command line input used to verify the model. Incidentally, this is of interest because it does not seem to require verifying separately all of the specifications, as demonstrated by the unreachable state results. One should also note that these commands need to be executed while in the working directory containing the Promela and C files.

Figure 11: Command Line Input



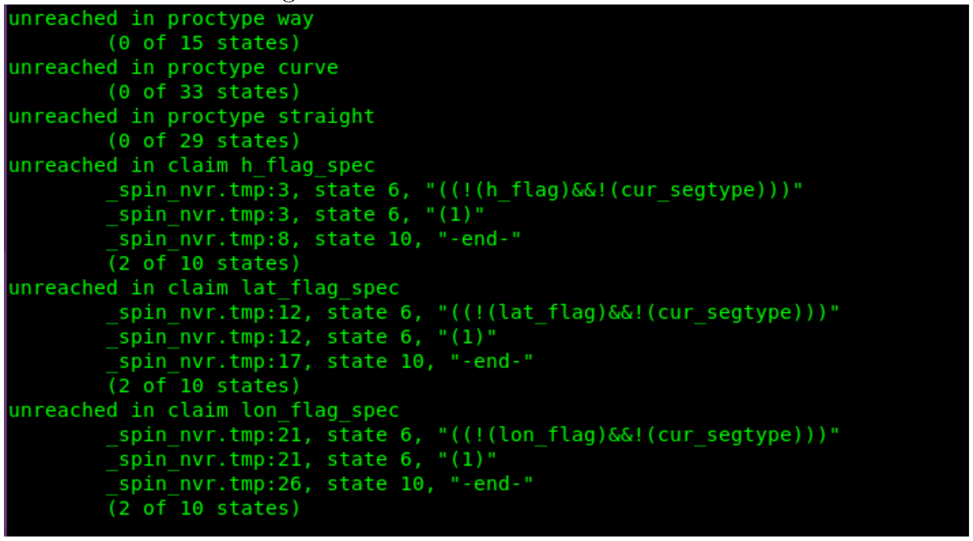
```
[mlmccle3@linuxremote2 AER-E-407-Final-Project]$ spin -a test.pml
ltl h_flag_spec: ! (<> ((! (h_flag)) && (! (cur_segtype))))
ltl lat_flag_spec: ! (<> ((! (lat_flag)) && (! (cur_segtype))))
ltl lon_flag_spec: ! (<> ((! (lon_flag)) && (! (cur_segtype))))
  the model contains 3 never claims: lon_flag_spec, lat_flag_spec, h_flag_spec
  only one claim is used in a verification run
  choose which one with ./pan -a -N name (defaults to -N h_flag_spec)
  or use e.g.: spin -search -ltl h_flag_spec test.pml
[mlmccle3@linuxremote2 AER-E-407-Final-Project]$ gcc -DMEMLIM=1030 -O2 -DXUSAFE -DNOCLAIM -lm -DBITSTATE -w -o pan pan.c
[mlmccle3@linuxremote2 AER-E-407-Final-Project]$ ./pan
```

The commands are also as follows here:

```
spin -a test.pml
gcc -DMEMLIM=1030 -D2 -DXUSAFE -DNOCLAIM -lm -DBITSTATE -w -o pan pan.c
./pan
```

The notable portion of the results is here:

Figure 12: Reachable State Results



```
unreached in proctype way
        (0 of 15 states)
unreached in proctype curve
        (0 of 33 states)
unreached in proctype straight
        (0 of 29 states)
unreached in claim h_flag_spec
        _spin_nvr.tmp:3, state 6, "((!(h_flag)&&!(cur_segtype)))"
        _spin_nvr.tmp:3, state 6, "(1)"
        _spin_nvr.tmp:8, state 10, "-end-"
        (2 of 10 states)
unreached in claim lat_flag_spec
        _spin_nvr.tmp:12, state 6, "((!(lat_flag)&&!(cur_segtype)))"
        _spin_nvr.tmp:12, state 6, "(1)"
        _spin_nvr.tmp:17, state 10, "-end-"
        (2 of 10 states)
unreached in claim lon_flag_spec
        _spin_nvr.tmp:21, state 6, "((!(lon_flag)&&!(cur_segtype)))"
        _spin_nvr.tmp:21, state 6, "(1)"
        _spin_nvr.tmp:26, state 10, "-end-"
        (2 of 10 states)
```

The major takeaway is that all of the possible states of the model are reached, but none of the states that would violate the specification are, meaning that the model and its specifications are ultimately successful.

6 Future Work

Given the success of this proof of concept and adequate resources and time, it is reasonable to assume the feasibility of similar verification efforts regarding the rest of the software suite. While other tools therein are significantly more complex, increased knowledge of model checking will allow me to expand my efforts.

There are also possible improvements that can be made to the straight flight path creation portion of the tool. The arbitrary approach to user input is a relic of the previous version of the tool, which was much more manual. I believe that, given the automated approach to the task, it may be more practical to anchor the beginning of each straight segment to the previous segment as it is handled in the curved segment portion. This may save user time and also help eliminate potential errors.

7 Bibliography

"Overview." Google Earth. Google. Accessed November 12, 2020. https://www.google.com/earth/.

Promela Reference – do(3), November 28, 2004. http://spinroot.com/spin/Man/do.html.

Promela Reference – ltl, September 6, 2017. http://spinroot.com/spin/Man/ltl.html.

Promela V4 Reference – c_code(7), November 28, 2004. http://spinroot.com/spin/Man/c_code.html.

Promela V4 Reference – c_decl(7), November 22, 2017. http://spinroot.com/spin/Man/c_decl.html.

"SPIN VERIFIER's ROADMAP: BUILDING AND VERIFYING Spin MODELS." Spin, December 3, 2010. http://www.spinroot.com/spin/Man/Roadmap.html.

WB-57 Experimenter's Handbook. NASA Johnson Space Center, June 2017. https://jsc-aircraft-ops.jsc.nasa.gov/assets/aod_338902.pdf.