Autonomous Cognitive Agent Flying Aircraft in X-Plane

Design Document

Team:

Avery Mackey

Brianna Marshall

Dylan Ostolaza

Zachary Reese

Faculty Advisor:

Dr. Siddhartha Bhattacharya

Client:

Dr. Siddhartha Bhattacharya

Table of Contents

1. Introduction 3
1.1. Purpose 3
1.2. Scope 3
1.3. Intended Audience 3
2. Definitions and Acronyms 4
3. System Design 5
3.1. Design Overview 5
3.2. System Architecture 5
4. Component Design 6
4.1. CCAT 6
4.2. Soar Agent 6
4.3. GUI 6
4.4. Datarefs 6
5. User Interface Design 7
5.1. Overview of User Interface Design 7
5.2. User Interface Screen Images 7
5.3. User Interface Objects and Actions 9
6. Appendix 9
6.1. Planned Additions to Existing Software 9
6.2. Current State of Agent 9

1. Introduction

1.1. Purpose

This is the design document for the Autonomous Cognitive Agent Flying Aircraft in X-Plane project. It outlines the overall architecture and design of the project, defining how the different software systems in the project interact with each other. Additionally, this document shows a visual of the current UI implementation, as well as how we intend to expand the UI to work with the improvements to the current system we are attempting to make.

1.2. Scope

This software is intended to be a SOAR Architecture agent that interacts with the X-Plane 12 flight simulator via the NASA Plugin XPC. The agent will be flying the Beta Technologies Alia-250 VTOL aircraft in X-Plane. The goal is to improve upon the existing agent that handles takeoff, landing, and flight to handle and react to various equipment failures that an aircraft could encounter mid-flight. This could mean diverting, or landing in a field even. The current agent handles single engine failure, and we plan on implementing a system to handle various sensor failures such as GPS or IMU failures. The agent currently does not support dynamically modifying its route mid-flight, so we plan on implementing this later as well to assist with engine failure management.

1.3. Intended Audience

The intended audience for this project is Dr. Siddhartha Bhattacharya, the aviation community, and software developers. This project will be a part of the 2026 Northrop Grumman Engineering and Design Showcase at the Florida Institute of Technology.

2. Definitions and Acronyms

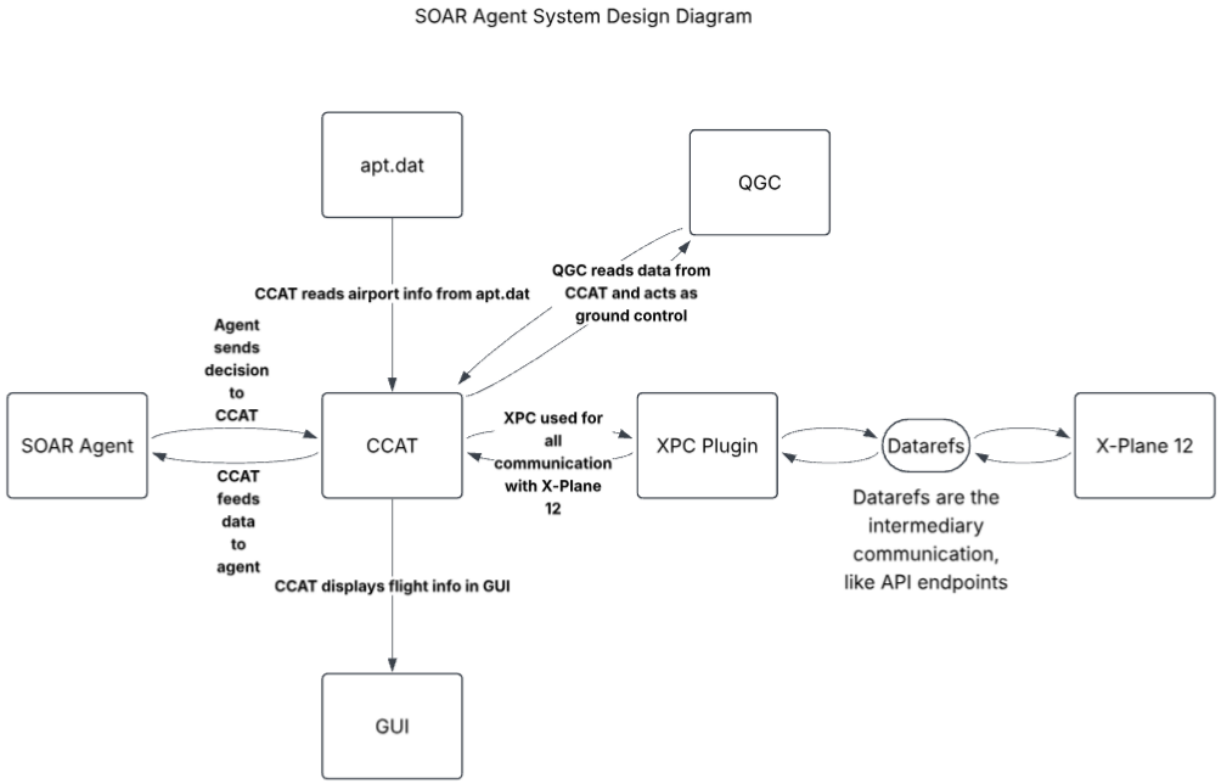
Agent: The SOAR program that determines the flight path for the Beta Technologies Alia-250 aircraft.
GUI (Graphical User Interface): A visual way to interact with computers utilizing graphics such as icons, buttons, and windows.
SOAR (State, Operate, and Result): A cognitive architecture for creating systems to exhibit human-like cognitive behavior.
XPC (X-Plane Connect): Open-source research tool developed by NASA that allows external programs to interact with the X-Plane 12 flight simulator.
X-Plane 12: Realistic flight simulation software.
Plugin: Software that adds existing feature(s) to an existing program.
FAA (Federal Aviation Administration): U.S. Government Agency responsible for safety of aviation in the United States.
QGC (QGround Control): Open-source ground control station for any MAVLink enabled vehicle.
MAVLink (Micro Air Vehicle Link): Lightweight communication protocol used to send data and commands between drones and ground control stations.
Beta Technologies ALIA-250: Electric VTOL aircraft designed for cargo, medical, military, and up to 5 passenger transport.
Autonomous: The ability of the agent to operate without human interaction or guidance.
Datarefs: Variables in X-Plane that show the internal state of various aspects of the flight simulator and aircraft.
CCAT: Java program harness for the SOAR agent and GUI
apt.dat file: Database file that contains all airports and their coordinates, runways, and info in X-Plane 12
Boolean: Descriptor for a statement meaning true or false.

3. System Design

3.1. Design Overview

The agent communicates with X-Plane 12 through the X-Plane Connect plugin to get details about the aircraft's flight through datarefs. XPC will send this data to the SOAR agent. The GUI is constantly updated with flight information. The user will trigger equipment and sensor failures through the GUI. The GUI will give this faulty sensor data to XPC, which will then be picked up by the agent. The agent will then process the flight data along with failures to determine the best course of action. The agent will then make its decision and send instructions back to XPC. XPC will then feed this information to X-Plane 12 where the aircraft flight is controlled by the agent input. The CCAT program reads from the apt.dat file to find suitable airports to divert to if necessary. The coordinates for these are passed to the agent for decision making, and the previous cycle repeats.

3.2. System Architecture



4. Component Design

4.1. CCAT

4.1.1. CCAT is the java program that harnesses the SOAR agent. It already exists from the existing project, and we are simply expanding it. It handles the high-level execution of the agent itself and all the interactions with XPC.

4.2. Soar Agent

4.2.1. The SOAR agent is the actual cognitive, autonomous agent that controls the aircraft in X-Plane 12. It handles all data processing and decision making for what happens to the aircraft in X-Plane. It will make the decision on what to do in the case of an equipment failure, and whether to divert. The SOAR agent is written using SOAR production rules that are read in by the CCAT Java harness program. SOAR production rules are boolean rules that the agent uses to determine its course of action.

4.3. GUI

4.3.1. The current GUI shows flight variables from X-Plane, and the status of the agent. It shows whether the agent is currently training or not. It allows for simulated engine failure. We plan on expanding the current GUI to have an array of equipment or sensor failure options for the agent to receive this input.

4.4. Datarefs

4.4.1. We plan to use various X-Plane 12 datarefs to provide the agent with the correct flight information. X-Plane has datarefs that allow for manipulation of flight data to simulate sensor data. The agent will read data from the cherry-picked datarers to determine the failure and react accordingly.

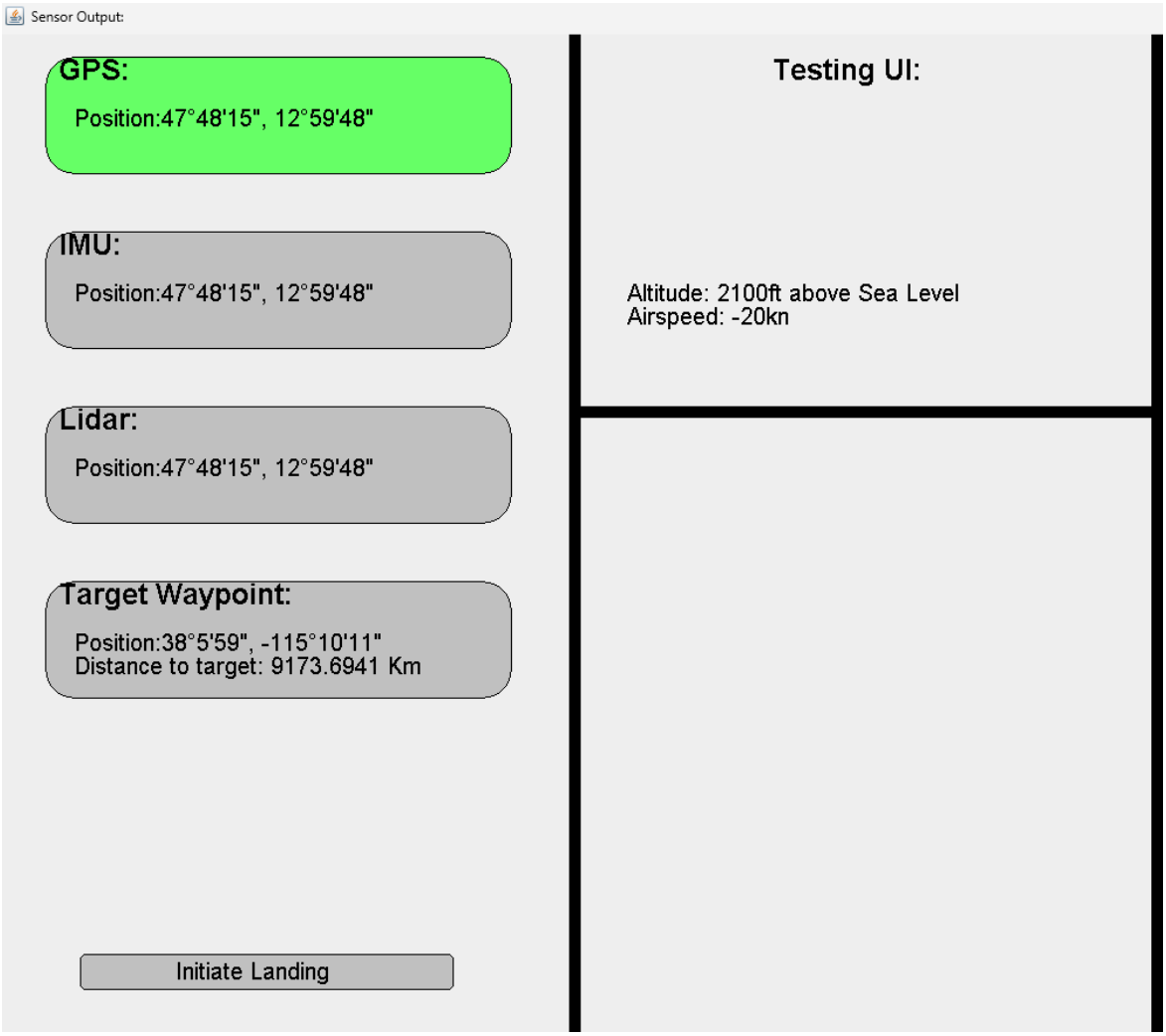
5. User Interface Design

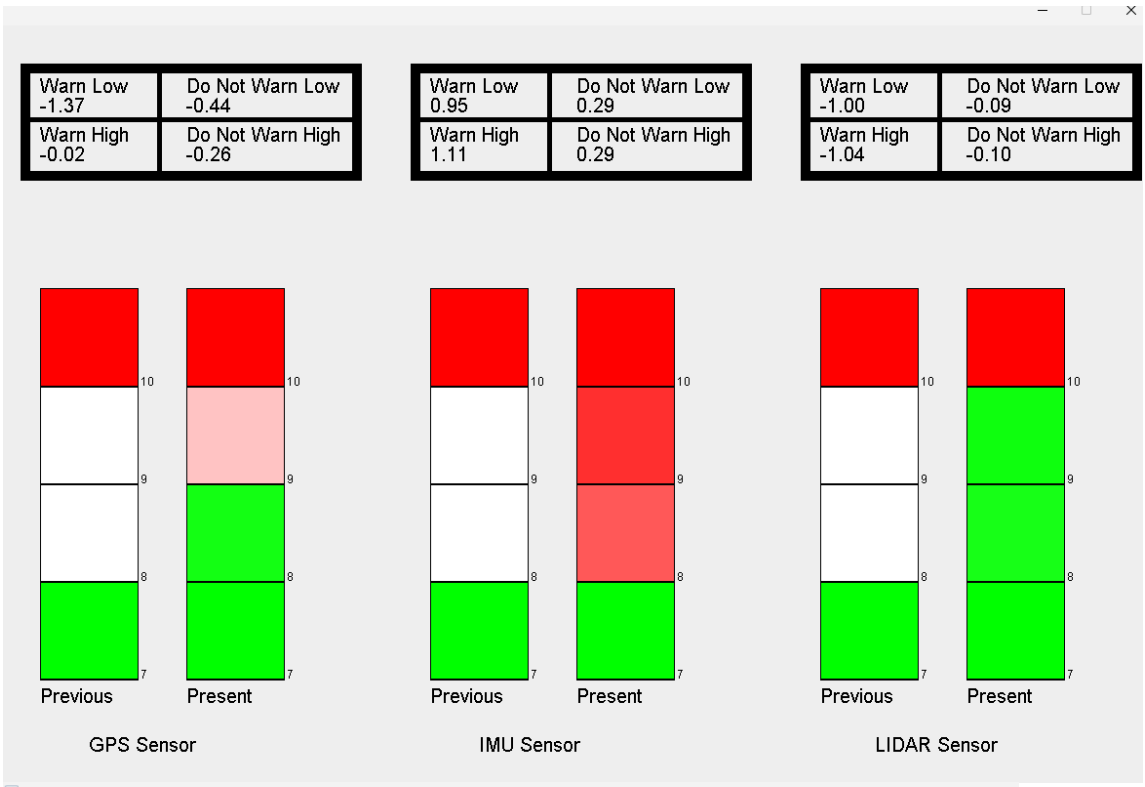
5.1. Overview of User Interface Design

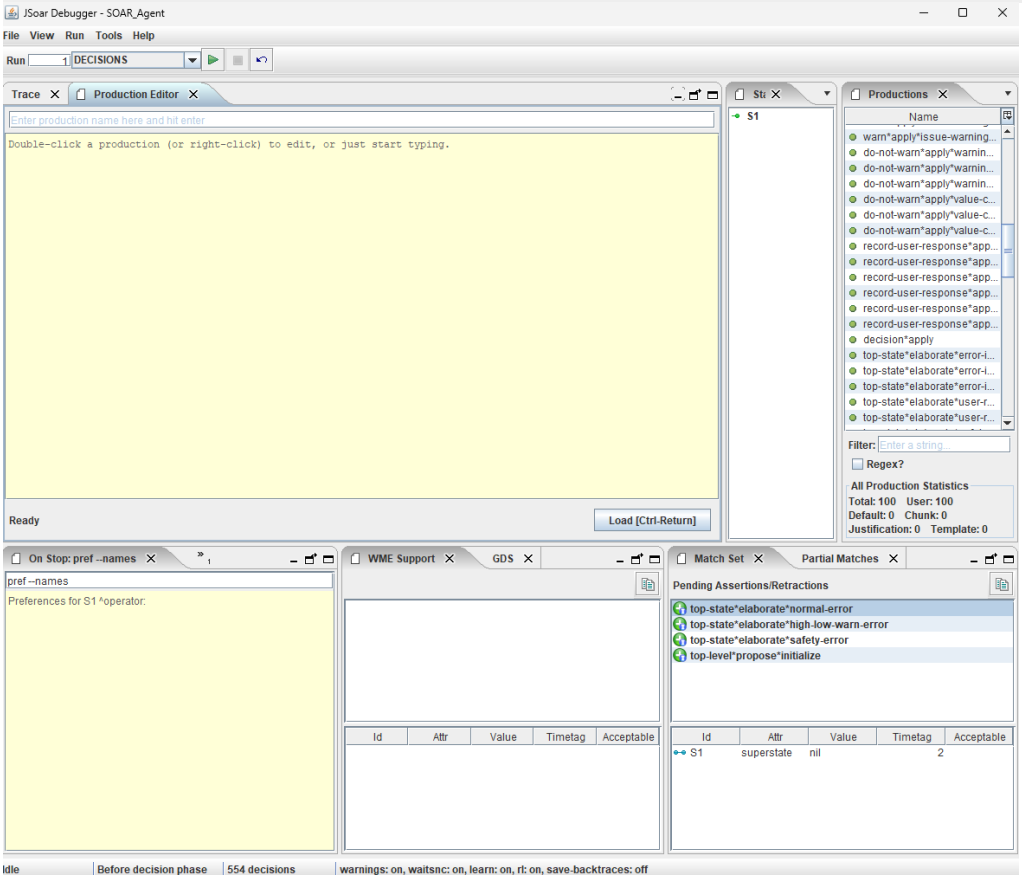
The CCAT user interface has already been designed to provide as much data to the end user regarding the agent as possible. The focus is on clear understanding and visibility of the data from the agent. We plan on expanding this to allow for simple simulation of sensor or equipment failures aboard the aircraft. It will consist of buttons for each failure in an easily readable area in the current GUI implementation. The GUI is relatively unimportant to the overall system, but it still has a purpose in the system.

As we continue to build on this system, more GUI components will be added for more equipment failures. The methodology for adding to the GUI will be maximum data visibility since the client is interested in the data and ability of the agent itself, rather than a modern UI.

5.2. User Interface Screen Images




Warn Low
-1.37
Do Not Warn Low
-0.44
Warn High
-0.02
Do Not Warn High
-0.26
Warn Low
0.95
Do Not Warn Low
0.29
Warn High
1.11
Do Not Warn High
0.29
Warn Low
-1.00
Do Not Warn Low
-0.09
Warn High
-1.04
Do Not Warn High
-0.10
10
9
8
7
Previous
Present
GPS Sensor
IMU Sensor
LIDAR Sensor


JSoar Debugger - SOAR_Agent
File View Run Tools Help
Run DECISIONS
Trace
Production Editor
Enter production name here and hit enter
Double-click a production (or right-click) to edit, or just start typing.
Ready
Load [Ctrl-Return]
S1
Productions
Name
Filter:
Regex?
All Production Statistics
Total: 100 User: 100
Default: 0 Chunk: 0
Justification: 0 Template: 0
On Stop: pref --names
pref --names
Preferences for S1 *operator:
WME Support
GDS
Id
Attr
Value
Timetag
Acceptable
Match Set
Partial Matches
Pending Assertions/Retractions
top-state*elaborate*normal-error
top-state*elaborate*high-low-warn-error
top-state*elaborate*safety-error
top-level*propose*initialize
S1 superstate nil 2
Idle
Before decision phase
554 decisions
warnings: on, waitsnc: on, learn: on, rl: on, save-backtraces: off

5.3. User Interface Objects and Actions

The first portion of the user interface shows the current sensor data. The agent currently relies on GPS, IMU, and Lidar for its navigation. It also shows the distance to its target waypoint, and a button to initiate a landing. The other window in the first image shows the UI for testing the agent. One can simulate errors with that portion.
The second image shows the status of agent training. It has different displays for the different values of the agent's variables and states for SOAR.
The final screen image shows the debugging window, which shows all the state transitions and more for the SOAR agent. It shows stack trace and more for simple debugging of the agent.

6. Appendix

6.1. Planned Additions to Existing Software

We are building on an existing system. Most of this architecture was already built before the initiation of this project. But the emphasis is not on building a new system, it is more on the AI side of this. Our goal is to build on this existing agent to handle multiple equipment failures and dynamically modify its flight plan mid-flight, which is beyond the agent's current capabilities.

6.2. Current State of Agent

The agent already has rules for landing and taking off, but it lacks a high-level controller to orchestrate the components. Our demo is a simple SOAR file that gets the agent to altitude and has it cruise to demo the system. We will utilize all the existing SOAR files in designing the new high-level .soar file that orchestrates the entire flight plan.