

Autonomous Cognitive Agent Flying Aircraft in X-Plane

Milestone 2 Progress Evaluation

Team Members:

Avery Mackey:	amackey2023@my.fit.edu
Dylan Ostolaza:	dostolaza2024@my.fit.edu
Zachary Reese:	zreese2023@my.fit.edu
Brianna Marshall:	bmarshall2022@my.fit.edu

Faculty Advisor:

Dr. Bhattacharyya: sbhattacharyya@fit.edu

Client:

Dr. Bhattacharyya, Department of Electrical Engineering and Computer Science:

Progress of current milestone:

Task: <i>Implement, test, and demo...</i>	Completion %	Avery Mackey	Dylan Ostolaza	Zachary Reese	Brianna Marshall	To do
Triggering and communicating with at least one aircraft equipment failure from X-Plane to the SOAR agent	100%	0%	0%	100%	0%	None
The agent's ability to	85%	0%	0%	20%	80%	Handle waypoint

follow a complete flight plan in the X-Plane simulation with no complications						s, update GUI to show actual nearest airports, update the GUI to show destination airport and any waypoints
SOAR rules that recognize the selected failure and determine an appropriate response based on the aircraft's remaining capabilities	100%	95%	0%	5%	0%	None
The agent issuing the appropriate commands in response to the failure	100%	0%	100%	0%	0%	None

Discussion of each accomplished task (and obstacles) for the current Milestone:

- Implement, test, and demo...
 - Triggering and communicating with at least one aircraft equipment failure from X-Plane to the SOAR agent

- Added a new window to the current testing window that allows for failures to be triggered
- Passes active failures into the SOAR kernel through the input link
 - `^io.input-link.flightdata`
 - `^allEnginesOK true|false`
 - `^failures`
 - `^active-count <int> ^last-injected <target>|none`
 - `^engine-0 ... ^engine-4 ok|failed (0-3 lift, 4 pusher)`
 - `^gps ^imu ^lidar ok|bias|drift|freeze|dropout`
 - `^altimeter ok|bias|drift|freeze|dropout`
 - `^radar-altimeter ok|bias|drift|freeze|dropout`
- Allows for a multitude of failures to be passed in:
 - Lifts 0-3 (being the quad rotors)
 - Pusher (5th rotor behind that propels the aircraft)
 - GPS
 - IMU
 - LIDAR
 - Altimeter for agent
 - This works by corrupting the altimeter data the agent receives
- These are passed through a new FailureInjector.java file
- Decided to add new UI window to trigger failures since current testing UI is mostly full, made more sense to have a smaller window to simulate all failures
- Different modes for sensor failures, being bias, drift, freeze, and dropout
 - Bias: constant offset error
 - Drift: grows steadily overtime
 - Freeze: reading stuck at current value
 - Dropout: data feed is lost entirely
- The agent's ability to follow a complete flight plan in the X-Plane simulation with no complications
 - The biggest part of completing this task was getting the agent to direct an aircraft through all phases of a flight on its own. The takeoff phase was completed during Milestone 1, but cruise, descent, and landing still needed to be added. This was done by adding sensor inputs for the bearing to the target destination, height

above ground, vertical speed, pitch, and ground speed. There were also commands added for pushing the throttle, toggling autopilot, and target heading implemented in Java, and flight logic for cruise, deceleration, and controlled vertical descent in Soar.

- Although a “full plan” with other waypoints has not yet been implemented, the agent can fly the aircraft straight to its destination, regardless of its departure. The agent decides how to navigate the trip using a new flight-phases.soar, which includes logic for cruising and vertical landing.
- There was a major issue with how the program communicates with X-Plane via XPlaneConnect. There was only one connection, shared by the entire program, with multiple threads attached to it. Although the threads run independently, their requests would overlap and get each other’s answers. For example, the aircraft is descending and is at 2000ft; the agent requests if it’s in the air (0) and receives a different reply for an altitude request (2000). Because the agent interprets any number above 0.5 as “on the ground,” the agent cuts the rotors and marks the flight as “landed” while it’s actually still 2000ft in the air. The overlapping replies were fixed, at least for now, by adding a lock to the X-Plane connection, which forces the threads to take turns on each request-reply.
- SOAR rules that recognize the selected failure and determine an appropriate response based on the aircraft’s remaining capabilities
 - This task was implemented in a new failure-handling.soar file (loaded through operators/load.soar) along with supporting guards in top-level.soar and flight-phases.soar. The rules monitor the ^io.input-link.flightdata.failures structure for any equipment or sensor target (engine-0 through engine-4, gps, imu, lidar, altimeter, and radar-altimeter) transitioning from ok to a fault mode (failed, bias, drift, freeze, or dropout) or returning back to ok. When a fault occurs, the agent records it on the state under ^recognized-failure and continuously evaluates the aircraft’s remaining capabilities under ^capabilities across four areas: vertical lift (full with 4 lift rotors working, degraded with 3 working, or critical with 2 or fewer working), forward pusher propulsion (ok or failed), position navigation (full, redundant, single-sensor, or lost, along with the highest-priority healthy sensor in ^preferred-nav-sensor), and altitude sensing (full, agl-only, baro-only, or lost).
 - Based on these remaining capabilities and the current flight phase, the agent selects a response (^failure-response, which is also mirrored to ^io.output-link.failure-response) and adapts how it flies the aircraft. If the active position sensor fails while a backup sensor is still healthy, the agent executes switch-position-sensor to switch ^sensor-to-use (gps to imu to lidar) and flags the faulty sensor as unreliable without interrupting the flight; if all three position sensors fail (navigation lost), it stops chasing corrupted bearings and

initiates a forced-vertical-landing in place. If the barometric altimeter fails, use-radar-altimeter falls back to ^agl to transition to cruise at 2,000 ft AGL, while a radar altimeter failure triggers cautious-vertical-descent to hold a gentle -150 fpm descent rate all the way to touchdown instead of relying on ^agl to slow down near the ground. For propulsion faults, any engine or critical sensor loss on the runway triggers abort-takeoff (cutting throttle to 0.0), while a failure during vertical climb triggers abort-climb-and-land to descend straight back to the ground. In horizontal cruise, losing the pusher propeller causes a forced-vertical-landing using the working lift rotors, and losing a single lift rotor triggers a precautionary-landing while boosting descent-throttle (0.75 to 0.90) and minimum lift limits (0.2 to 0.45) so the remaining three rotors compensate for the lost thrust. Conversely, if two or more lift rotors fail in cruise while the pusher propeller still works (lift critical, pusher ok), the agent recognizes that the aircraft can no longer hover and selects maintain-wing-borne-cruise—shutting down the asymmetric lift rotors, canceling any hover deceleration, and keeping the autopilot and pusher propeller active so the aircraft stays wing-borne above stall speed (or entering an emergency-descent at full remaining lift if the pusher also fails).

- Two main obstacles came up while integrating these rules with the existing agent. First, because failures can be injected on the exact same decision cycle as normal flight transitions (begin-cruise, begin-landing, start-vertical-descent) or sensor warning rules (warn, record-user-response), having multiple operators proposed with best (>) preferences caused operator tie impasses in SOAR; this was resolved by adding explicit operator precedence rules so touchdown and pilot aborts take top priority, followed by failure recognition, failure responses, normal flight transitions, and finally continuous control and alert rules. Second, the existing approach logic in flight-phases.soar schedules target airspeed (70 kt down to 35 kt to 0 kt) based on distance to the destination waypoint, which meant an in-place failure landing triggered far from the destination would keep flying at 70 kt instead of stopping. Adding an ^emergency-land yes flag allowed failure-triggered landings to override the distance schedule and immediately command ^target-airspeed 0, while splitting output-link apply rules so they only replace values that actually differ prevented SOAR from deleting output attributes when re-applying the same command value.
- The agent issuing the appropriate commands in response to the failure
 - Added a new FailureResponseHandler.java that receives the response SOAR selects, waits for the commands to settle, then reads lift and pusher throttle, autopilot state, and the active position sensor back from X-Plane and logs a PASS or FAIL for each check

to the console and to failure-response-log.csv for the test plan.

- The checks run on their own thread so the agent loop is not blocked, and the log shows which responses were verified and which have no X-Plane-observable effect.
- The main difficulty was proving a response actually worked, since the agent sending a command does not mean the aircraft carried it out. Each response had to be matched to values X-Plane could report back, and some responses have no visible effect in X-Plane, so they are logged as informational only.

Discussion of the contribution of each team member to the current Milestone:

- Avery Mackey
 - Implemented the SOAR rules in failure-handling.soar (along with updates to flight-phases.soar and top-level.soar) that recognize active equipment and sensor failures from the SOAR input link and determine an appropriate response based on the aircraft's remaining capabilities. The rules evaluate remaining lift rotors, the pusher motor, position sensors (GPS, IMU, LiDAR), and altimeters, then select and issue the matching response—such as switching to a healthy backup sensor, maintaining wing-borne cruise when vertical lift is lost, or executing an immediate precautionary or emergency landing with boosted lift throttle on the remaining rotors. The difficult part was preventing SOAR operator ties when failures occurred simultaneously with normal flight-phase transitions and overriding the normal distance-based airspeed schedule so the agent could slow down and land in place rather than waiting until it reached the destination.
- Dylan Ostolaza
 - Implemented the Java side of the agent's failure responses: a FailureResponseHandler that receives the response SOAR selects, reads the aircraft state back from X-Plane, and logs PASS/FAIL results to a CSV for the test plan. Checks cover lift and pusher throttle, autopilot state, and the active position sensor. The difficult part was proving a command had worked, since the agent sending a command doesn't mean the aircraft did it.
- Zachary Reese
 - Implemented new FailureInjector.java for passing various aircraft equipment failures to the SOAR agent. Added new failure window for tester to simulate these and pass this to the agent. 4 failure modes for

sensors, described above. Each rotor can be toggled on and off, including the pusher motor (propels the beta alia 250 from the back of the plane). The difficult part was finding a singular route to pass all of this to the SOAR agent, so it is not looking in multiple places at once for an error. Equipment failures can be read through the SOAR input link described above. Made a demo video of the different equipment failures being triggered and XPlane showing the aircraft responding.

- Brianna Marshall
 - Implemented missing logic in the Java application to automate controls for the aircraft and added the soar decisions for flight navigation. Made the communication between X-Plane and the Java application more functional by converting headings to magnetic, automating autopilot button presses, and locking requests to X-Plane. Updated the “Initiate landing” button in the GUI to force a landing wherever the aircraft currently is, instead of at its destination. Made a demo video of the aircraft automatically completing a flight with no user input

Plan for the next Milestone:

Task: <i>Implement, test, and demo...</i>	Avery Mackey	Dylan Ostolaza	Zachary Reese	Brianna Marshall
Automate the flight planning for a normal operation flight	10%	10%	10%	70%
The agent responding to multiple component failures in parallel	70%	10%	10%	10%
A complete flight plan with one or more equipment failures	10%	70%	10%	10%
Milestone 2 cleanup: Altitude, landing, GUI updates, FAA verification	10%	10%	70%	10%

Discussion of each planned task for the next Milestone:

- Automate the flight planning for a normal operation flight

- Use an API to retrieve the entire flight plan, including waypoints, from the departure (read from X-Plane) to the destination (provided by the user). Have the aircraft fly this entire flight plan, with hit waypoints being shown on the GUI for verification.
- Implement, test, and demo the agent responding to multiple component failures in parallel
 - Implement, test, and demo the agent responding to multiple component failures in parallel. Extend the SOAR failure-handling rules and Java response handler so the agent can detect and respond to multiple simultaneous equipment and sensor failures within a single flight rather than handling only one failure at a time. Update the agent's capability evaluation and operator preferences to resolve compounded failures—such as a position sensor or altimeter failure occurring alongside one or more rotor failures—by applying compatible fallback actions in parallel (e.g., switching to a backup sensor while simultaneously boosting lift throttle and initiating a precautionary landing). Test and demonstrate these multi-failure scenarios in X-Plane to verify that all required commands are issued and executed without conflict.
- Implement, test, and demo a complete flight plan with one or more equipment failures.
 - For the next milestone, I will develop a complete flight plan for our autonomous aircraft in X-Plane that includes one or more equipment failures. I will define the flight route, establish the flight parameters, and determine where and how the failures will be introduced. I will also work on the expected responses of the autonomous cognitive agent, ensuring that it can recognize the failures, evaluate the situation, and take appropriate actions to maintain safe flight. The goal is to test the agent's ability to handle unexpected situations while completing the flight plan.
- Milestone 2 cleanup: Altitude, landing, GUI updates, FAA verification
 - Fine-tune and improve the implementations made for milestone 2. Improve the agent's altitude control to maintain a more consistent cruising altitude at the beginning of the flight phase; reduce the number of hard landings the aircraft makes; update the GUI to display failure solutions and flight planning; verify that the failures align with FAA standards.

Date(s) of meeting(s) with Client during the current Milestone:

- None

Client/Faculty Advisor feedback on each task for the current Milestone:

- Triggering and communicating with at least one aircraft equipment failure from X-Plane to the SOAR agent
 -
- The agent's ability to follow a complete flight plan in the X-Plane simulation with no complications
 -
- SOAR rules that recognize the selected failure and determine an appropriate response based on the aircraft's remaining capabilities
 -
- The agent issuing the appropriate commands in response to the failure
 -

Faculty Advisor Signature:_____

Date:_____

Evaluation by Faculty Advisor

Faculty Advisor: Detach and return this page to Dr. Chan (HC 209) or email the scores to pkc@cs.fit.edu

Score (0-10) for each member: circle a score (or circle two adjacent scores for .25 or write down a real number between 0 and 10)

Avery Mackey	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10
Dylan Ostolaza	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10
Zachary Reese	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10
Brianna Marshall	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10

Faculty Advisor Signature:_____

Date:_____