Communication, Navigation and Control using FlightGear Simulations

Aero 450: Flight Software Systems
December 10, 2012

Duncan Miller
Hrishi Shelar
Joshua Thomas

Introduction & Motivation

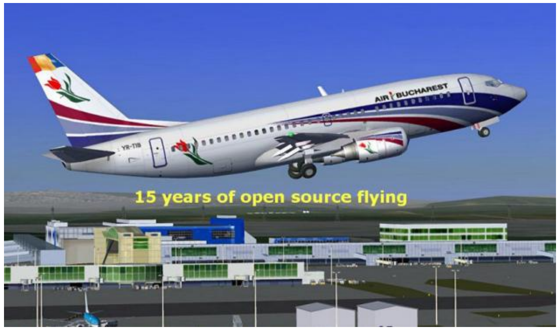
- Autonomous, unmanned aerial vehicles (UAVs) are beginning to operate regularly in the National Airspace System
- There is an increasing need to test the coordination and control of flight vehicles to optimize operations and flight paths
- An autonomous vehicle testbed would allow sophisticated testing of control algorithms in a protected, repeatable environment

Project Proposal

- Design an aircraft controller that will generate different flight patterns based on ground object movement
  - Use simulated input and the FlightGear environment to solve real-world applications
  - Build on open source software as much as possible
- Three main building blocks for the project:
  1. Inner loop stabilization
  2. Basic Maneuvering
  3. Advanced Flight Path

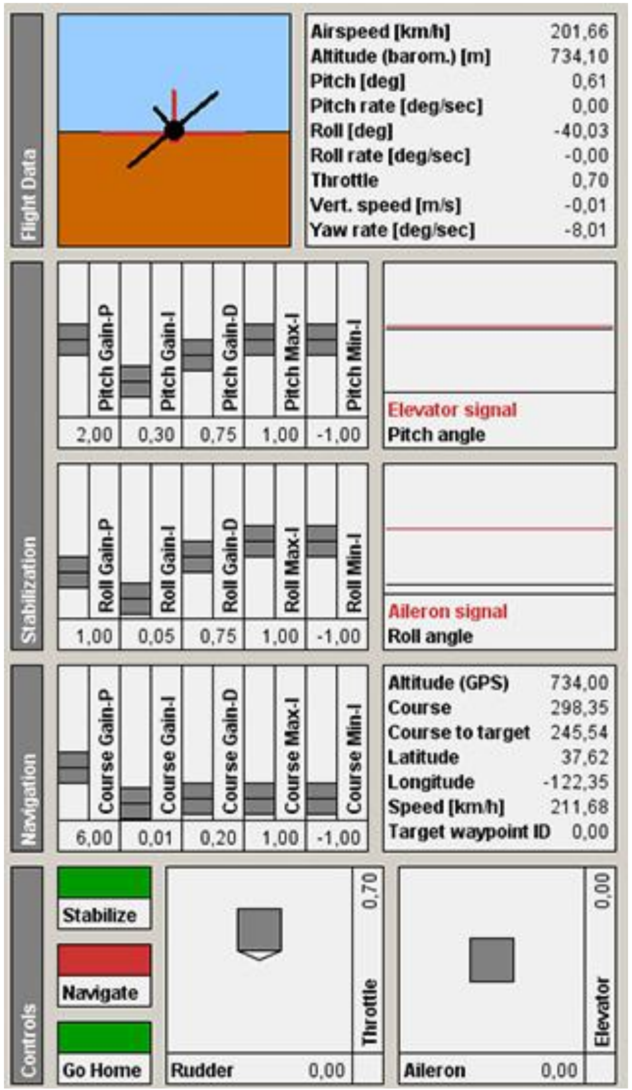
What is FlightGear?

- Free open-source flight simulator development project
- 3D rendering of aircraft and surrounding area
  - Realistic→updates with local weather!
- Outputs: position, roll, pitch, yaw, airspeed, angular velocities
- Inputs: flap deflections, throttle



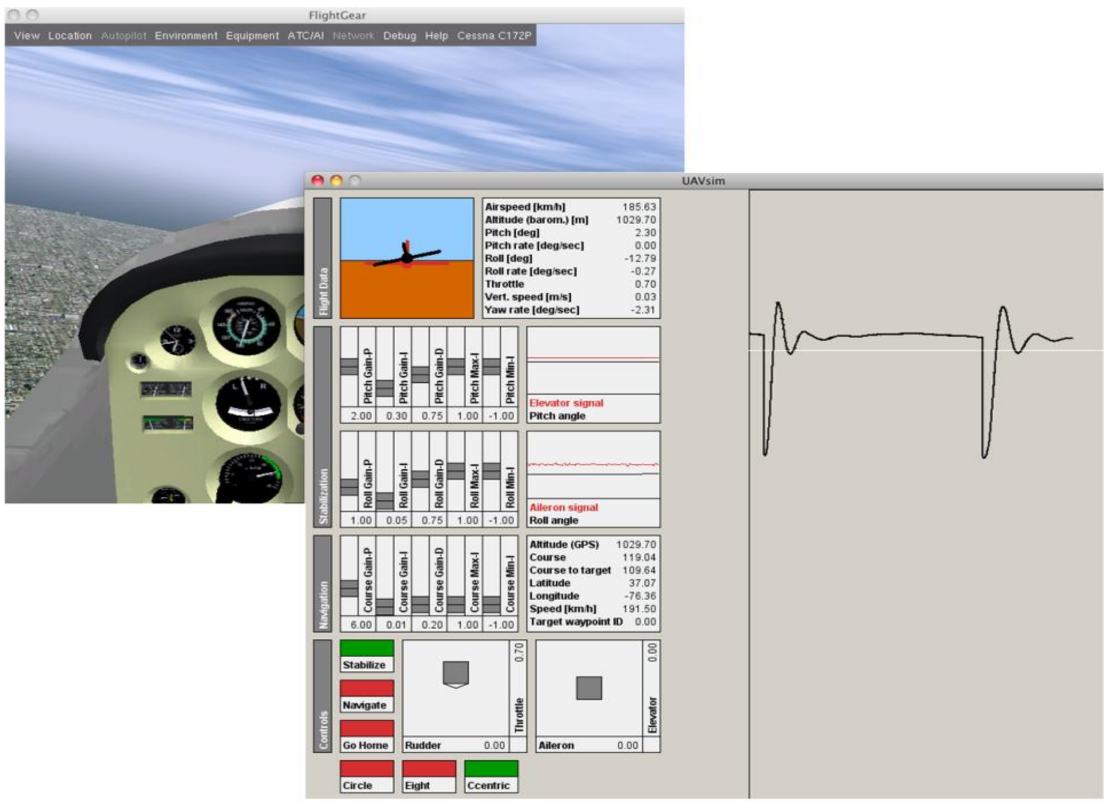
UAV Playground

- Open source Java application
- Enables us to begin testing flight algorithms immediately
  - ✓FlightGear socket communication
  - ✓Inner-loop stabilization
  - ✓Data and gps logging



Preliminary Augmentations

- Real-time error plotting
- Three new flight modes: looping, concentric circling, figure-8



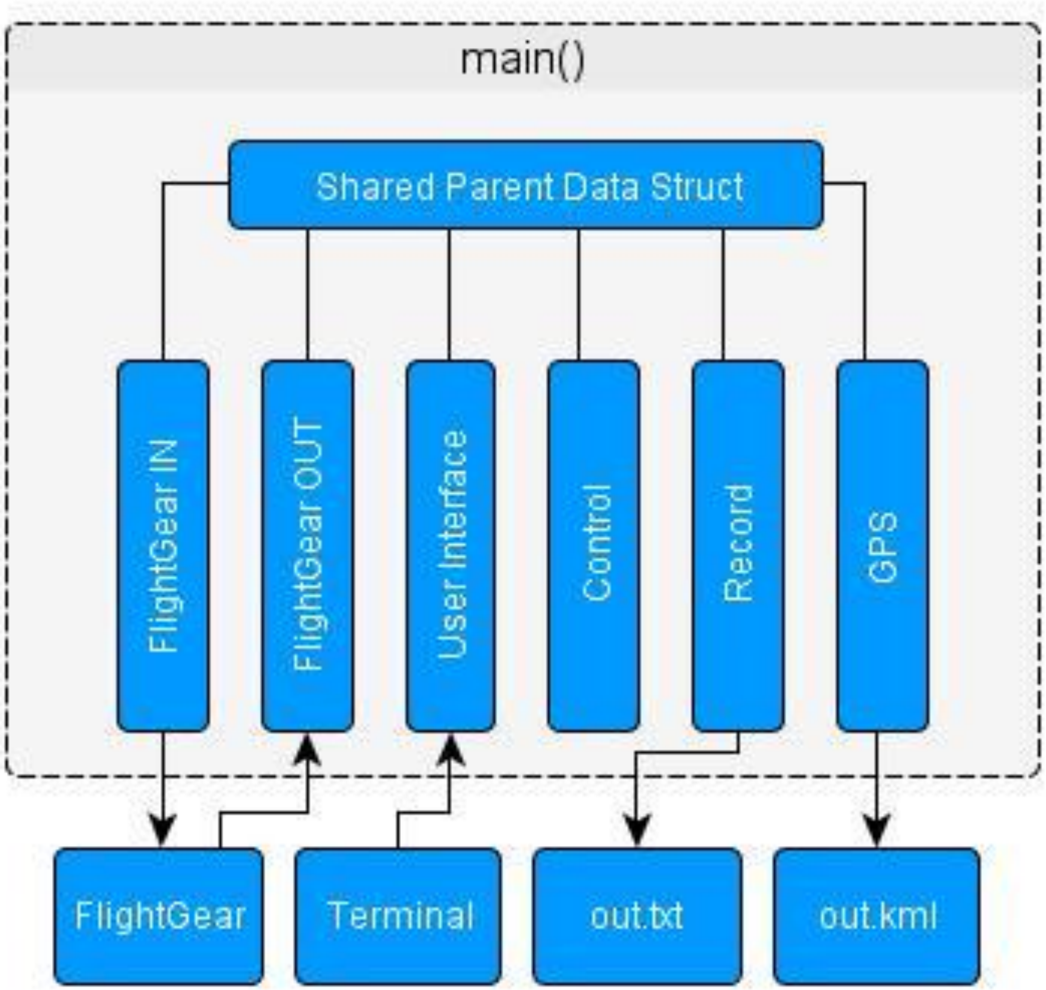
Memory Leaks

- The “platform independent” language (Java) perhaps not so platform independent
- Our team needed to learn how to create and compile projects in both Eclipse and the Processing IDE
- UAV Playground proven to operate on Mac OS
- In both Linux and Windows, program crashed after 30 seconds of runtime
  - Suspected memory leak still an open question

Transition to C++

- Reasons for transition:
  - Unable to resolve memory overflow of UAV Playground
  - More satisfying constructing a functioning program from scratch
  - More experience with the C++ functions presented in class (sockets, threads)
- Resulted in project de-scope:
  - The stabilization functionality, originally black-boxed and considered complete, had to be re-created in C++
  - Basic maneuvering (steady level, turning and climbing) completed
  - Applications of flight path not considered

Threaded Architecture



Data Structure and Data Sharing

- The ‘parent’ data struct is shared between all threads

Header (.h)

```
struct UIs {
        char mode;
        double value;
        bool record;
};

struct parent {
        double fg_in[13];
        double fg_out[4];
        UIs UI;
};
```

Main (.cpp)

```
int main() {
        pthread_t receive;
        pthread_t input;
        pthread_t control;
        pthread_t transmit;
        pthread_t record;

        parent data;

        iret1 = pthread_create(&receive, NULL,
        &receivef, (void *) &data);

        iret2 = pthread_create(&input, NULL,
        &inputf, (void *) &data);
```

Etc…

Socket Communication

- Use of UDP protocol over sockets to establish link between FlightGear and the program
  - Sockets on the same machine
  - No error checking, hence faster
- Three sockets
  - FlightGear out: Telemetry such as altitude, heading, speed, etc.
  - FlightGear in: Commands for throttle, aileron, elevator and rudder
  - NMEA out: GPS location

Communication Protocol

- Protocol specified by XML files placed in FlightGear program folder
- FlightGear out generated XML file that contained 17 telemetry fields (mimicked UAV playground protocol)
- FlightGear in received commanded values separated by tabs (mimicked UAV playground protocol)
- NMEA out generated NMEA sentences (simulated a GPS module)

Data Sharing

- Telemetry data, Command data and User Input stored in common structure
- Passed to each thread
- Every read/write function on common data is protected by mutexs
- Three Mutexs
  - OutMutex
  - InMutex
  - UIMutex

Bugs and Difficulties

- Used Simple_Sock functions given to us by Prof. Atkins
- Buffer size that held incoming socket data too small
- Buffer size that held outgoing socket data too big
- Troubleshooting:
  - FlightGear interpreted newline as ‘\r\n’ as compared to ‘\n’
  - Negative values caused buffer size to change because of ‘-’ sign

Buffers and Data Parsing

- Input data stored to buffer than passed to parse function
- Self-written state machine to parse XML telemetry data.
  - Worked by counting ‘>’
- Output data taken from shared data structure.
  - Four float values, separated by tabs. End with ‘\r\n’
  - sprintf(buff, "%1.3f\t%1.3f\t%1.3f\t%1.3f\r\n", data_r->fg_out[0],data_r->fg_out[1],data_r->fg_out[2],data_r->fg_out[3]);
  - Socket buffer size changed with each command depending on number of ‘-’ signs

Data Logging

- Data logged at a rate of 20 Hz
- Both Telemetry and Commanded values
- Mutex locking while reading in values from the shared data structure
- User option to start/stop logging
- Stored in ".txt" file that can be imported into Matlab, etc for post processing

GPS and KML

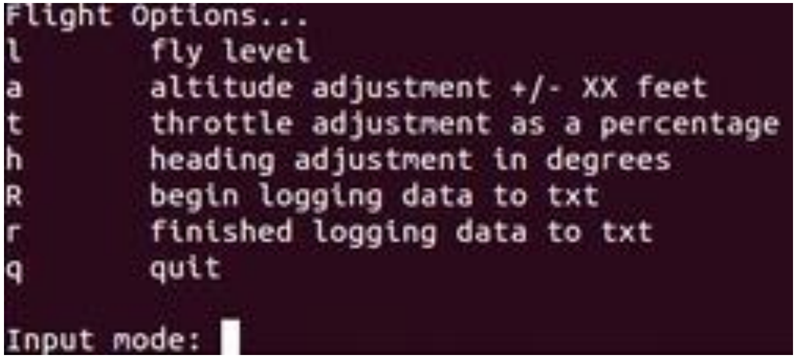
- NMEA is standard GPS module output format
- $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
- Time, latitude, longitude, and altitude information
- Parser used state machine to check for $GPGGA then count commas
- Google Earth KML file is type of XML file
- KML file updated at 1 Hz with new fix data

Google Earth Plot

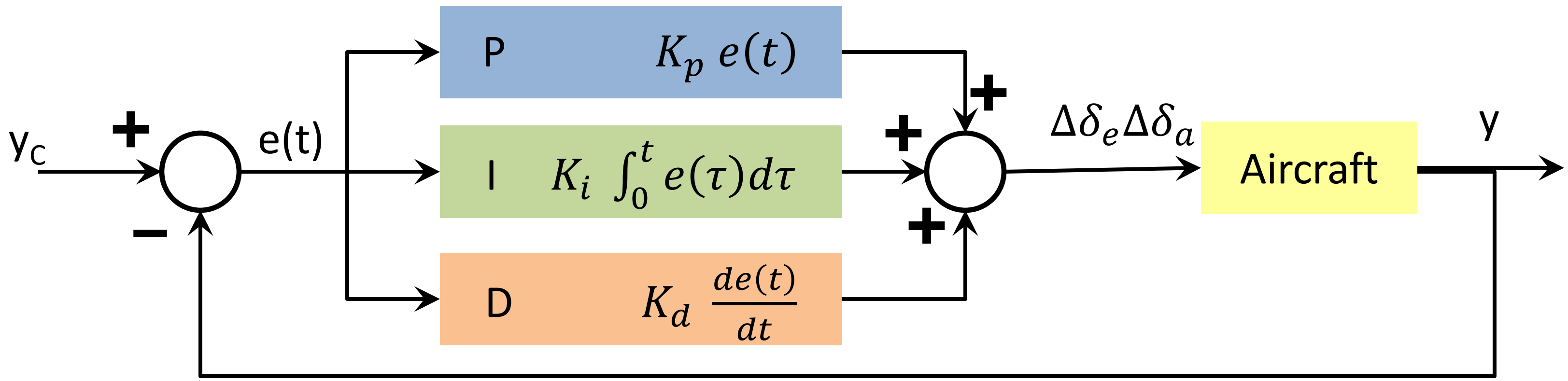
KML File read by Google Earth

User input

- Input thread takes in user commands from the keyboard
- Fly level 'l'
  - Program stabilizes the plane into a safe, level state
- Change altitude 'a'
  - User inputs desired altitude and plane pitches up/down to achieve desired state
- Change heading 'h'
  - User inputs desired compass heading and plane banks to achieve desired state
- Quit 'q'
  - Program returns control of the plane to the user



PID Control



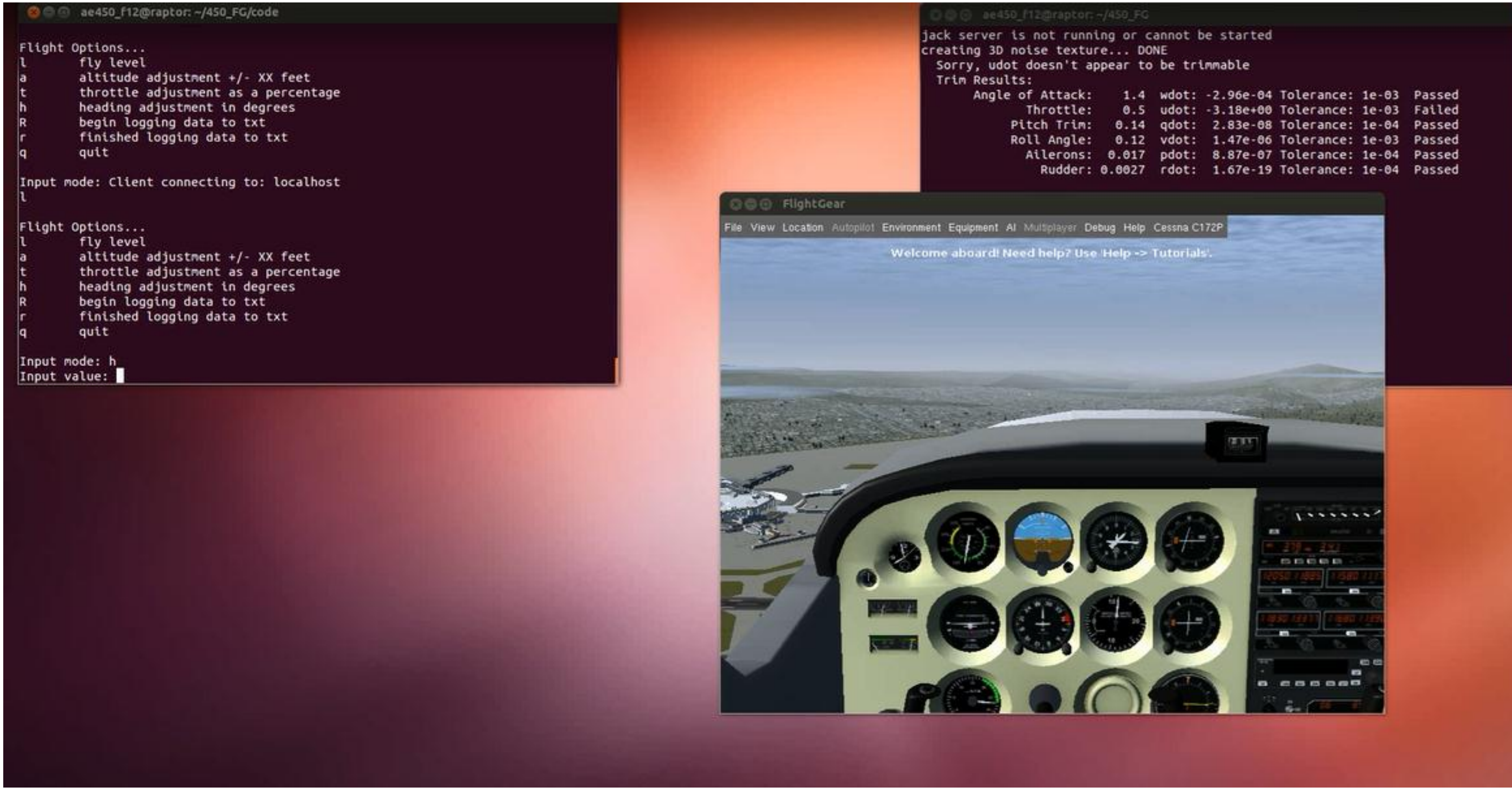
- Standard PID control feedback
- Rudder $(\Delta\delta_r)$ *is not being used*
- Pitch is controlled with a P controller
- Roll is controlled with a PID controller
- Current Gains used

  K_p = 0.02
  K_d = 0.01
  K_i = 0.01

Modes and parameters

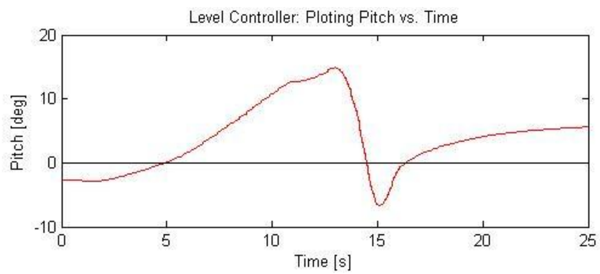
- Samples current state at 10 Hz and feeds error into PID control function
- Commands flaps to desired position
- Fly level ‘l’
  - Commands roll and pitch to zero
- Change altitude ‘a’
  - Commands pitch to +/- 10˚
  - Levels out when within 50ft of desired altitude
- Change heading ‘h’
  - Commands roll to +/- 20˚
  - Switches to mode ‘l’ when within 5˚ of desired heading
- Quit ‘q’
  - Returns no commanded control to FlightGear

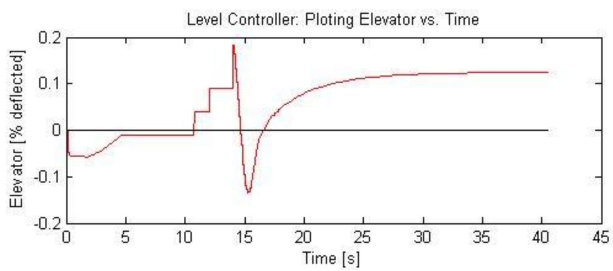
Videos – Stabilization and Turning

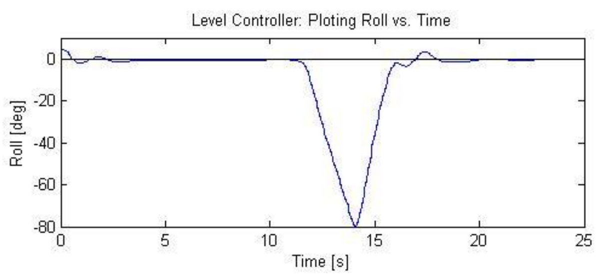


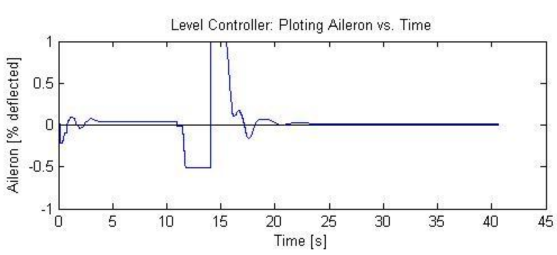
Plots and Results - Level Flight

P=0.02 I=0.01 D=0.01

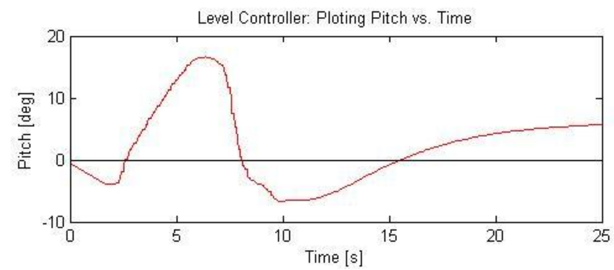


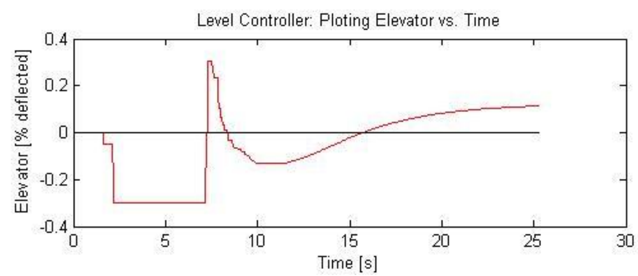


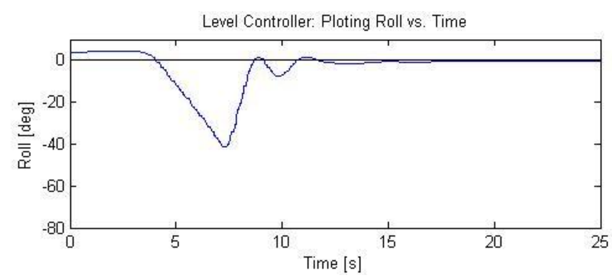


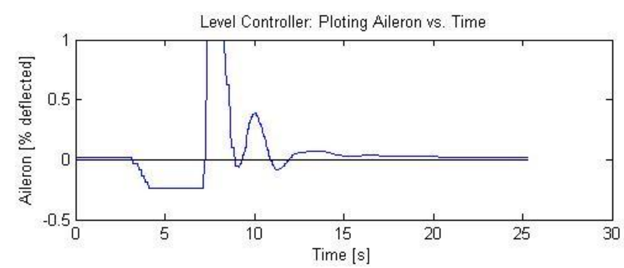


P=0.02 I=0.0 D=0.01

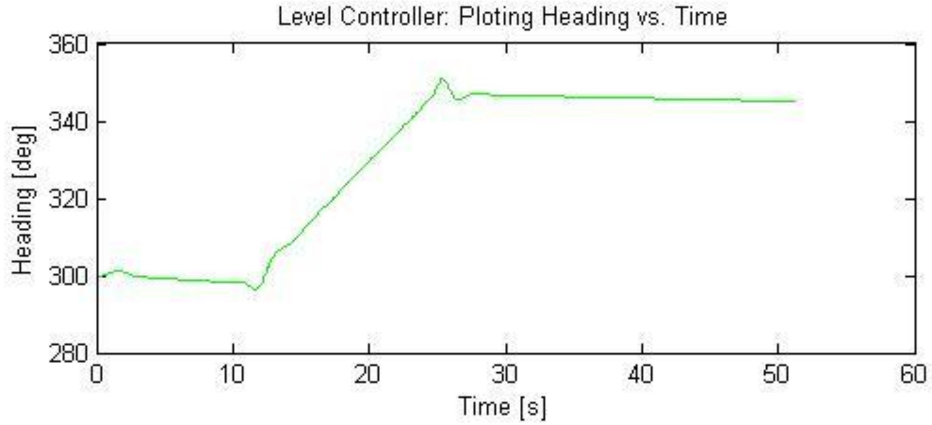


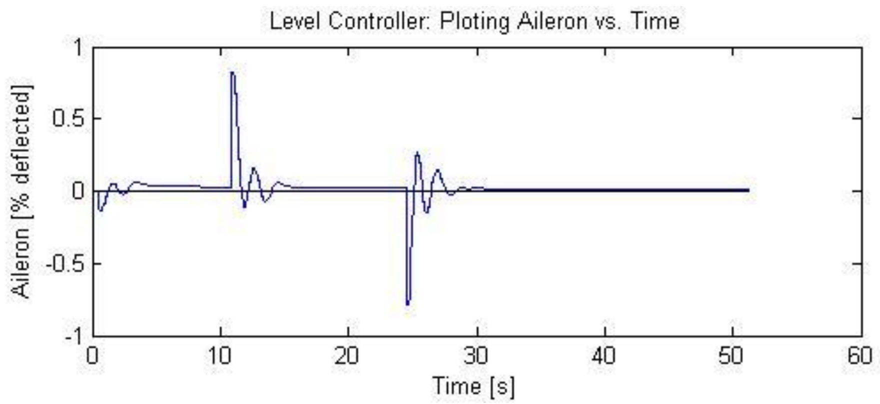






Plots and Results – Turning Flight





Skills and Lessons Learned

- Learned and applied the Java language and Processing IDE to create/augment a GUI
- Learned to compile projects and libraries in Eclipse
- Successfully demonstrated socket communication in a runtime environment
- Implemented PID controllers in roll, pitch and yaw for a commanded flight mode
- Gained experience with advanced C++ capabilities (multi-threaded code)

Future Work

Before semester end:

- Optimize specific gains for roll/pitch/yaw
- Comment code
- Additional error plotting and GPS paths in Google Earth
- Write final report

Potential testbed applications:

- Continue on original proposed project plan
  - Ground vehicle tracking (loiter, protect, survey)
- Simultaneous control of multiple flight vehicles
  - In-flight refueling algorithms

Back-up Slides

- Expanded control system (original)