Autopilot for an Ultra-Light, Flying Wing

Nashlie H. Sephus
Computer Engineering
Mississippi State University
nhs7@msstate.edu

Graduate Mentor: **Todd Templeton**
Research Supervisor: **Dr. Jonathan Sprinkle**
Faculty Mentor: **Prof. S. Shankar Sastry**

August 4, 2006

Summer Undergraduate Program in
Engineering Research at Berkeley (SUPERB) 2006

Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley

Autopilot for an Ultra-Light, Flying Wing

Nashlie H. Sephus

Abstract

The purpose of this project is to develop an auto-pilot for a small, light fixed-wing aircraft named the Zagi. This aircraft is interesting because it is inexpensive, simple and fast to deploy, and is virtually indestructible since it is made of expanded polypropylene (EPP) foam. This aircraft is also challenging from a control perspective because it is vulnerable to wind and can only carry a minimal payload. First, we develop optimal local trajectories given current wind conditions. These local trajectories are used to determine the path that the vehicle should try to maintain between widely-spaced waypoints, and trade off between overshooting corners and maintaining the desired trajectory. The testing and results of this portion are performed in MATLAB. We then implement these local trajectories in the Zagi autopilot (written in C) to enable it to follow an incrementally-specified global trajectory with future planning. Initial tests are performed using the CRRCsim simulator interfaced to the Zagi hardware. Results from these tests show that the planning autopilot is better than the existing autopilot which verifies our invitation that planning with three points at a time is superior to planning with only one point. Our results show that this higher level planner can be implemented in a greedy fashion with little difference in the path planning result (and large improvements in runtime). Future work involves final testing on a real Zagi at Richmond Field Station.

1 Introduction

Autopilots are useful in systems such as formation flight, automated air-traffic control systems, and satellite constellations. These systems are comprised of many similar units that interact directly with their neighbors and that have sensing and actuating capabilities at every unit. In this research project, we restrict our attention to the Zagi[5], which is a small, fixed-wing aircraft (see Fig. 1). We use the Zagi because it is light, easy to deploy, and virtually indestructible since it is made of strong, expanded polypropylene (EPP) foam; it is interesting from a control perspective because it is vulnerable to wind and can only carry a minimal payload.

The existing Zagi autopilot[4] from Crossbow plans for a single waypoint and a given nominal speed. We want to use this autopilot to follow higher level three-point trajectories, where the first and third points define vector approach and departure, respectively, from the second point. Planning for three points at a time is intuitively better because the three points represent the essential local information required for planning an entire desired trajectory. The planner would not gain much if more than three points were used because the three-point planner can choose new sets of points at any time. For example, the given three points $x1, x2, x3$ can be changed sometime between $x2$ and $x3$, or possibly before $x2$ if the high-level planner changes its mind. This higher level controller is implemented on the same microprocessor as the low-level controller in C. The entire system consists of the modified Crossbow autopilot and sensor pack, which is attached to the Zagi and controlled by a radio controller (RC), the Crossbow ground station[3], which is where the waypoints are given, and the CRRCsim[1] Zagi simulator for the hardware-in-the-loop (HWIL) testing (see Fig. 1).

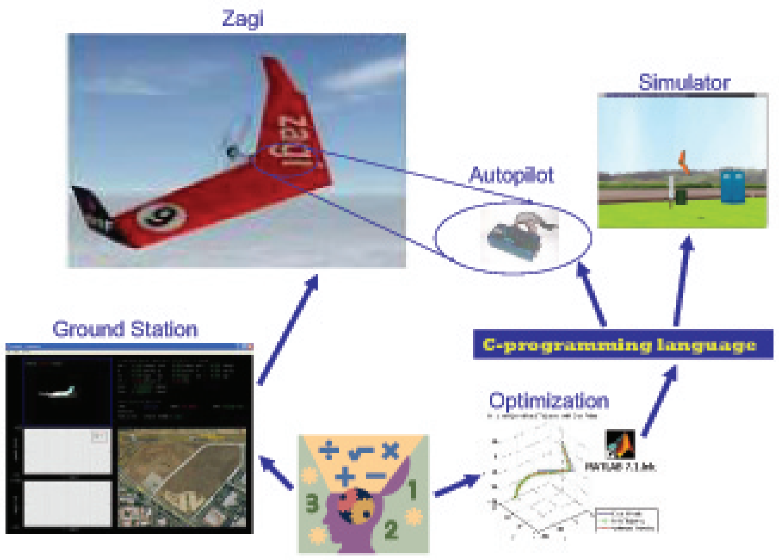

Figure 1. Project outline demonstration.

2 Formulation of the Problem

Specifically, the first step is to develop local optimal trajectories given three points at a time. These trajectories are used to determine the path that the vehicle should try to maintain between widely-spaced waypoints by trading off between overshooting corners and staying on the desired lines as long as possible (see Fig. 2). The planner first creates an initial (greedy) trajectory that satisfies the speed, velocity, acceleration, and turning rate constraints based on the initial point, initial velocity, and three waypoints given. This initial trajectory is then submitted for local optimization.

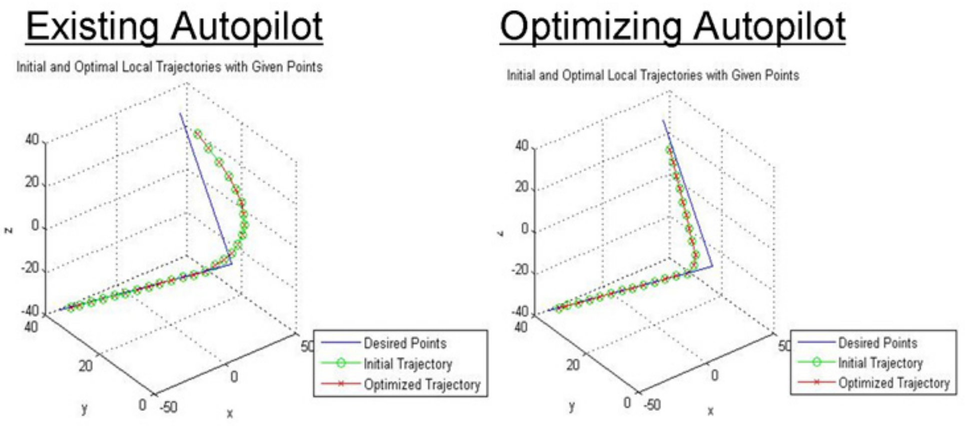

Figure 2. Comparing autopilot trajectory paths.

2.1 Creating Initial Trajectory

The initial trajectory must account for speed, velocity, acceleration, and turning rates. Given initial point $p0$, three waypoints $p1$,$p2$,$p3$, and initial velocity, $v0$, where $l12$ and $l23$ represent the lines between points $p1$, $p2$ and points $p2$,$p3$, respectfully (see Fig. 3):

```
While current point $p$ in trajectory is not within close range to $p3$,
    Check distances from $p$ to $l23$.
        If $p$ is closer to $l12$, check distance from $p$ to $p2$.
            If $p$ is within close range to $p2$, new point $p$ approaches $l23$.
            Else, $p$ is not close to $p2$. Find closest point $pc$ on $l2$ to $p$ and approach $pc$.
        Else, $p$ is closer to $l23$. Find closest point $pc$ on $l23$ to $p$ and approaches $pc$.
    Check orientation (roll, pitch, yaw)[2] to ensure constraints are not violated.
        Maintain maximum turning rates (angular velocity) if possible, else increase as much as allowed.
    Check speed to ensure constraints are not violated.
        Maintain maximum speed if possible, else increase or decrease toward maximum speed as much as
        the maximum acceleration will allow.
    Update initial trajectory to add the new point $p$.
    Check distance from $p$ to $p3$, which determines whether or not to loop again.
end loop
```

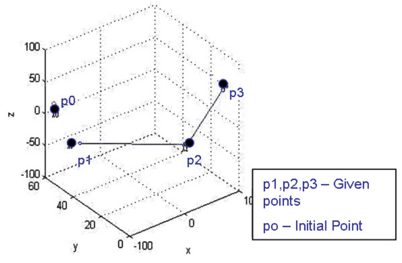

Figure 3. Three given waypoints and initial point of aircraft.

2.2 Algorithm for Local Optimality

Local optimization is based on the following optimality criterion, where t and $l(t)$ represent the point and line, respectfully at time t, $l12$ and $l23$ represent the lines between points $p1$, $p2$ and points $p2$,$p3$, respectfully, and *gamma* is a positive constant:

$$\min \sum_{0}^{t_{max}-1} \left\{ \min \left[dist\left(p\left(t\right), l_{12}\left(t\right)\right), dist\left(p\left(t\right), l_{23}\left(t\right)\right)\right] + \gamma \times dist\left(p\left(t\right), p_3\right)\right\} \tag{1}$$

Our cost function does the following:

```
Check constraints on all speeds in the current trajectory.
    If any speed is greater than maximum speed, return large error.
Check constraints on all acceleration in the current trajectory.
    If any acceleration is greater than maximum acceleration, return large error.
Check constraints on all turning rates in the current trajectory.
    If any element of angular velocity (differences in pitch and yaw) is greater than maximum angular
    velocity, return large error.
Evaluate equation (1) and return resulting cost.
```

3 Implementation And Testing

3.1 Implementation in MATLAB

Our first prototype is implemented in MATLAB. We wrote a custom plotting function which allowed us to view the trajectories at each iteration and created movies of the optimization as it progressed. For the local optimization function, we used MATLAB's *fminsearch* function. *fminsearch* finds the minimum of a scalar function of several variables, starting at an initial estimate (unconstrained nonlinear optimization). For example, *x = fminsearch(fun,x0)* starts at the point x0 and finds a local minimum *x* of the function *fun*.

3.2 Simulation Results

During the final testing, runtime was always less than one minute for approximately 10 to 100 points on the trajectory. Test cases include both large and small distances between waypoints, obtuse and acute angles between waypoints, various positions of the initial point (see Fig. 5 and Fig. 6), and different initial velocity directions (see Fig. 4). The physical constraints had to satisfy the vehicle and operate efficiently with all test cases. The best values for constraints and other variables that were sensible for all test cases, as well as physically realistic for the vehicle, were the following:

```
maximum angular velocity (pitch and yaw) = 30 degrees/sec in each direction
maximum acceleration = 1m/s^{2}
maximum speed = 5m/s
gamma constant = 10
initial velocity = (4m/s to 5m/s)
```

As shown in the resulting figures, both the initial and the local optimal trajectories are identical for almost all test cases. Therefore, our greedy function for creating an initial trajectory is sufficient, and the local minimizing functionality is not necessary.

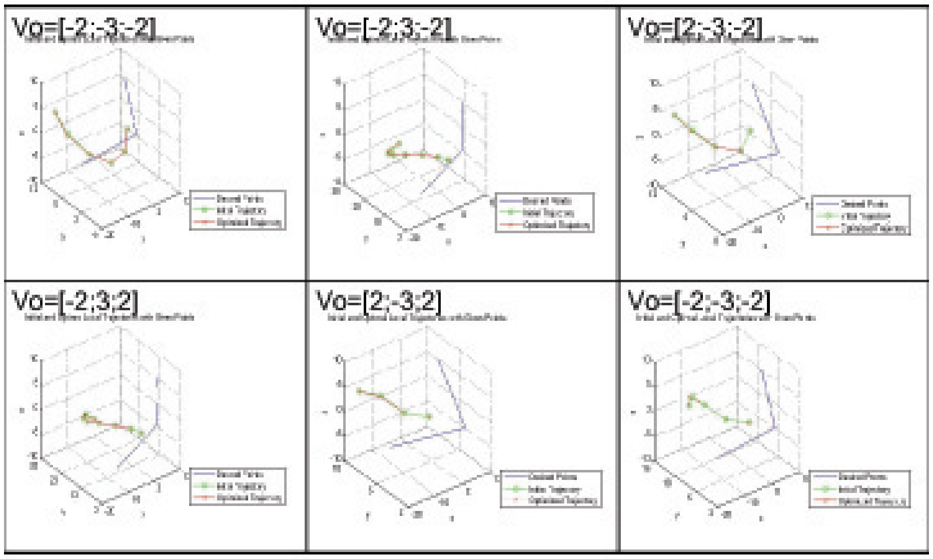

Figure 4. Same test case with varying initial velocities.

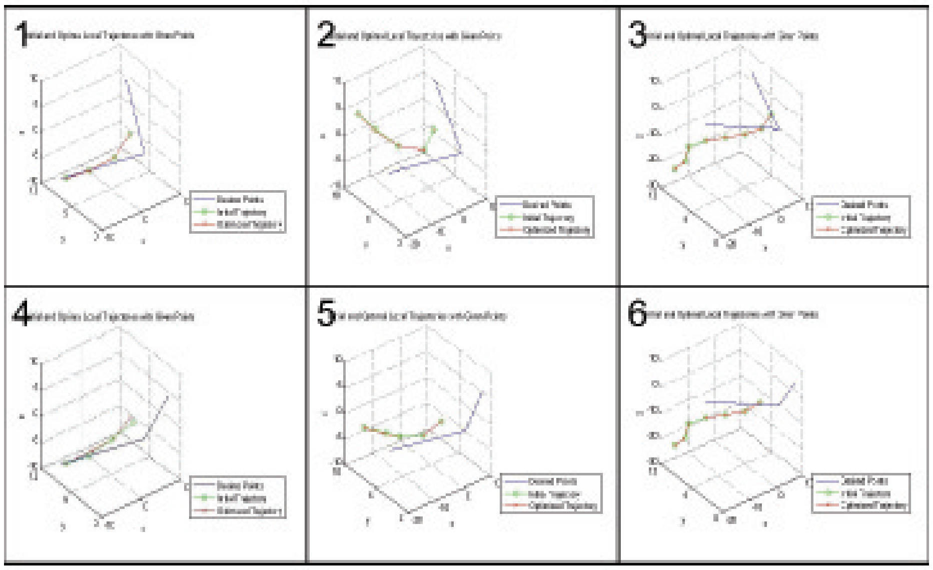

Figure 5. Test cases 1 - 6, small distances between waypoints (5m-16m).

3.3 Integration And Testing With Existing Autopilot

Since the existing autopilot was written in C, the MATLAB code was manually ported to C. This process was greatly simplified by the observation that we do not need to perform the local minimization. Few changes to the existing autopilot were necessary to integrate the code. Initial testing is performed using the CRRCsim simulator interfaced to the Zagi's autopilot hardware, which is connected to the RC. The RC switches are toggled to activate the autopilot and the three-point planner. The CRRCsim simulator displays the Zagi's movement in 3D, which operates on a Linux OS laptop, while the waypoints are inputed from the ground station, which operates on a Windows OS laptop. Also, the ground station displays the vehicle's movement from an aerial view (refer to Fig. 1). The combined system was demonstrated at the SUPERB final poster session[1]. Final testing of the full autopilot will be performed on a real Zagi at Richmond Field Station (RFS).

4 Conclusion

The new high level planner for this autopilot accepts an initial point, three waypoints, and an initial velocity, which allows planning to create a better trajectory than a system that only considers one point at a time. Also, physical constraints of the aircraft such as maximum values of velocity, acceleration, and orientation are enforced for more realistic planning. The planner was originally developed to create an initial trajectory that would be submitted to a local optimizing function that would minimize the cost between the initial trajectory and the desired waypoint trajectory. However, we demonstrated that the initial and local optimal trajectories are nearly identical, so the greedy algorithm for creating the initial trajectory is sufficient by itself. Therefore, the final implementation has less complicated code and shorter runtime (without the cost of optimization). Future work for this project includes further testing with the simulator, testing with a real Zagi, and including wind compensation.

Acknowledgements

Special thanks to Todd Templeton for his guidance, motivation, and patience throughout the summer. Thanks to Dr. Jonathan Sprinkle and Professor Shankar Sastry for their consideration and helpfulness. Also, I appreciate NSF and Intel for funding me this summer. I thank all of CHESS and SUPERB for this great summer experience. Last, but not least, I especially appreciate the love and support from my family and friends.

References

[1] Crrcsim simulator. http://crrcsim.sourceforge.net/.

[2] Euler zyx (roll, pitch, yaw). http://web.mit.edu/2.05/www/Handout/HO2.PDF.

[3] Ground station on xbow website. http://www.xbow.com/Products/productsdetails.aspx?sid=133.

[4] Zagi autopilot code. http://sourceforge.net/projects/micronav.

[5] Zagi website. http://www.zagi.com.

[1]August 4, 2006 at Wozniak Lounge in Cory Hall, UC Berkeley

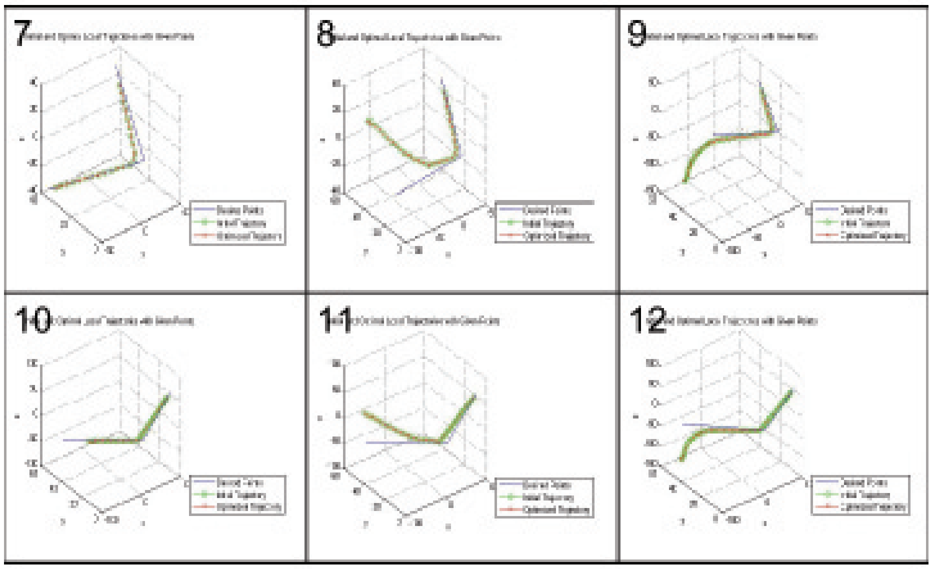

Figure 6. Test cases 7 - 12, larger distances between waypoints (55m-105m).