University of Advancing Technology

Project Aeolus

An Autonomous Lander for High Altitude Payloads

UAT SPACE

Brandon Michelsen

About

Project Aeolus serves as an autonomous landing system for high altitude balloon payloads. The system uses a parafoil and two stepper motors to allow for controlled descent of the payload as soon as the balloon pops. The software for the system is written using the Arduino codebase. Arduino provides the libraries necessary for interfacing with the hardware. The system is controlled through the ATSAMD21, a 32-bit MCU based on the Cortex-M0+. For updates on the project, see the Gitlab page: [<https://gitlab.com/brandonmiche/project-aeolus>]

Table of Contents

About………………………………………………………………………………………………i

Table of Contents………………………………………………………………………………….ii

Introduction………………………………………………………………………………………..1

Hardware Requirements…………………………………………………………………………...2

Software Requirements……………………………………………………………………………4

Software Design…………………………………………………………………………………...6

Hardware Design………………………………………………………………………………...10

Initial Prototype………………………………………………………………………………….11

Introduction

Project Aeolus is a system of hardware and software that allows payloads from high altitude balloons to guide themselves back to the ground. The platform is fed the GPS coordinates of possible landing locations before the launch. After the balloon reaches its highest altitude and pops, the system will release a drogue chute and will go into free fall. Once the platform reaches an altitude of 35,000 feet, it will release the main parachute. The system will then identify the nearest landing point an begin heading for it, using the onboard sensors for navigation. The platform will continue moving down until it finally reaches the landing point or reaches the ground.

The software for the platform is written using the Arduino codebase. Arduino provides a wealth of libraries for interfacing with the hardware used in various commercial and open-source flight controllers. The libraries allow access to sensors and controller functions, making development of flight control firmware easier.

The current choice for the flight controller is a custom board based on the ATSAMD21 MCU, a 32-bit microcontroller based on the ARM Cortex-M0+. However, due to the generic development of the Arduino platform, there are many other choices for a flight controller if the need for a different platform arises.

Hardware Requirements

This section lists the hardware requirements for the project. The hardware required for the project involves several sensors, a servo motor, stepper motors, and the flight computer. As stated above, the current choice for the flight controller is a custom board. However, if the need arises for a different controller, the Arduino codebase is able to support a switch.

Below is the list of requirements for the hardware:

* Flight Computer
* ATSAMD21 MCU
* STM LSM9DS1 IMU (compass, accel., gyro.)
* MS5611 Barometer (onboard)
* GPS Module
* SD Card Connector
* Radio Telemetry
* Bluetooth Connectivity
* Mechanics
* 9 Gram High Power Servo Motor
* Micro Stepper Motors
* Servo Arms
* Power
* 3.7 Volt 2000 mAh LiPo Battery

The hardware needs to be able to sense various features of the landing vehicles attitude – that is, it needs to sense the position in space, rotation in space, heading, and altitude. Sensing these features will allow the flight computer to successfully control the platform while it is in the sky.

In addition, we want to be able to send commands to the flight computer to tell it where safe landing locations are as well as other mission commands. There are two ways the board will allow us to do this. Firstly, the ATSAMD21 supports USB communication, which will allow us to send commands to the flight computer over USB. Secondly, the flight computer will support Bluetooth Low Energy (BLE) which will allow a Bluetooth application to send mission commands to the device.

Software Requirements

This section lists out the requirements for the software of the project. Much of the software involves interfacing with sensors and determining the platforms state. From there, the platform can set control signals to begin moving toward the drop location. Below is the list of required functions the software needs to perform:

* Sensors
* Determine angular rate of change
* Determine relative heading
* Determine acceleration
* Determine global position
* Determine attitude from sensed values
* Telemetry
* Receive mission commands from devices
* Data Collection
* Collect data from sensors
* Store collected data on an SD card
* Motors
* Generate stepper and servo motor control signals
* Control
* Determine the nearest landing location
* Use determined attitude to generate control signals for moving to target landing location
* Generate control adjustments to keep a stable attitude

The main purpose of the software is to control the descent of the payload and guide it back toward a safe landing location. It will accomplish this task by comparing the current location of the payload with the location of the landing location. Comparing these two positions will give us the relative bearing between the two, which we can compare with the heading of our vehicle. This comparison will give us an error value, which we can feed into a PID controller that will adjust the control on a parachute. This will allow the craft to steer back to a safe landing location (ignoring wind drift).

Software Design

The design of the software for the autonomous lander involves a high-speed control loop, in which the sensor values are measured, the error between the heading of the craft and the relative bearing of the given landing location is calculated, and a control value is produced to correct for the error. This control value can then be sent to the stepper motors to adjust the parachute accordingly.

The flight computer must also be able to receive Bluetooth and USB commands, as well as store data from the flight (i.e. attitude, GPS coordinates, etc.) in the SD card. The Bluetooth/USB commands must be detected ad received before the flight, as this will allow the user to program in the desired GPS coordinates of the landing location. Uploading the flight data to the SD card will happen at the very end of the flight (once the device has landed).

A basic diagram of the code is given below:

A close up of a map

Description automatically generated

This flowchart shows the path that the control loop will need to follow in order for the flight computer to successfully run. The control loop begins by first checking if Bluetooth or USB is connected. If it is, it pauses normal functionality to receive commands, after which it will resume normal operation. Otherwise, it continues through with normal execution.

Next, the flight computer will check if it has entered a fall. If it has not, it keeps checking. If it has entered a fall, it will begin in-flight operation. The in-flight operations involve checking if the device has landed, measuring sensor values, storing those sensor values, calculating an error with those sensor values, and then correcting for that error. This loop continues until the device has landed.

Once the device has landed, it will move the gathered data to the SD card. The data will be stored as a CSV file. After the data has been loaded onto the SD card, the program will end.

The error value can be calculated as the difference between the device’s heading (relative to Magnetic North) and the relative bearing between the device’s GPS location and the GPS location of the landing site. The flight computer can use this difference in the control loop to adjust for any error.

The control loop acts as a closed-loop control system, that constantly adjusts for error by feeding a PID output signal to each stepper motor. This will control the parachute, aiming it toward the landing site. A diagram of the control loop can be seen below:

A screenshot of a cell phone

Description automatically generated

Hardware Design

The hardware design is rather simple. It involves the flight computer microcontroller, the ATSAMD21, the sensor module, the Bluetooth module, the power module, the USB and programming modules, and any interface electronics. The interface electronics involve mostly passive electronic components necessary for the other modules to operate.

Below is a simple diagram showing the connections between the ATSAMD21 and the other modules. Detailed schematics will be drawn based on this module diagram.

A close up of a device

Description automatically generated

Initial Prototype

As testing the initial design with a payload falling from 100,000 feet in the air may lead to undesirable results, a ground-based prototype is needed. The ground-based prototype will need two parts: the test payload and a launching mechanism.

The test payload will consist of a hollowed-out sphere. The sphere will be shaped like a golf ball, allowing for aerodynamic rotation during the initial flight. The sphere will be cut into two parts, a top half and a bottom half. These halves will be held together by a rubber band, which can be released by a servo-controlled pin. The pin will only release when parachute is ready to deploy.

Inside the sphere, the control electronics, motors, and parachute will be housed. These will all sit in the bottom half of the sphere. The parachute will be attached to the control stepper motors via a two drums, one for each motor. The flight computer will sit in the middle, along with all the necessary sensors.

The launching mechanism will consist of a pressurized tube that can launch the payload. There will be a tube for the payload to sit in, and a pressure chamber that will launch the payload. A basic PVC air cannon would work well for this.

The initial tests will involve launching the payload from the launch mechanism and testing the flight control loop. The flight computer will detect when it has been launched, wait a few seconds, and then release the parachute. After the parachute has been deployed, the computer will enter the control loop. From there, the behavior of the device can be observed and documented, and flight control data can be gathered.