G(A)SP Balloon Simulation

Languages: Python, Arduino (C++ modified)

Designers: Maddie Mackey, Andrew Wilkie

Category: Home Automation

Requires: Python 2.7.1; Pygame 1.9.1 (at least)

The G(A)SP Balloon simulation models the conditions and descent of the G(A)SP high altitude weather balloon after the release of a steerable parachute over time, planned to be launched by the mechatronics class in September 2016. The model runs using Python 2.7.1, enabling pygame (which executes python modules for graphics and simplified GUI), and self-constructed physics, GUI and simulation files.

Balloon Simulator

The Balloon simulator is designed to simulate everything our stratospheric balloon will experience. the simulator will allow us to test and develop control algorithms and gain information about what the balloon and payload might go through during the journey. The simulator consist of two main parts, the control algorithms and the simulation itself.

The control algorithm will mainly control the parachute, which will indirectly affect the turning of the craft.

# Physics

This simulation required the implementation of physics into python code. Already knowing many of the equations we needed, we decided to write our own physics file to perform these equations to our needs, rather than to source and import a physics library or physics engine. After much research into the effect of altitude on the physical components of the parachute, this enabled us to better sculpt a file specifically to suit our needs – containing variables directly related to the descent. Functions (such as air\_density(), pressure() and temperature()) allow us to determine these variables for further functions that directly affect the simulation (such as drag() and glide\_angle()) at certain altitudes.

# Simulation

The simulation.py file contains the core of the simulation itself: containing the data for the balloon, calculating how the parachute must steer against the imported physics values in order to get home (the PID algorithm (calculatePID()))and updating the graphics to the GUI.   
It would also receive the data from serial when using a live feed of data to run the simulator.

# Libraries

All pygame libraries are included.

The built-in python libraries used are:   
- Math  
- Random  
- Time

# Instructions

## Configuration File

The configuration file (config.txt) contains variables that the user can alter to suit their purposes, such as the width and height of the GUI, the starting location of the balloon’s descent and the location of home, and environmental factors such as wind speed and direction. Including this turns the simulation from just an app for our personal purposes, but into one that can be easily altered by users with different intentions.

The simulation then is used by running the simulation.py file.

# Serial

Would allow us to:

* Read and stream live data

**#Write here if you want to talk about serial else delete title and get Maddie to change it**  
- How it relates it to the physical world, if it would read in data. And therefore relates it to home automation, as it completes the system: generating input from the physical world using sensors, receiving it through serial and sent into the user interface (the simulation), to then calculate the output and sent back to actuators (the steerable parachute) altering the next input. This creates a feedback loop.

# Controller

The steerable parachute attached to the balloon’s payload would serve as the controller to this system, if it were to receive live data.

* Arduino
* Run on the Edison
* Send data

#WHAT WE’RE ACTUALLY SIMULATING.

# Equations

The equations used that effect the simulation are:

## Newton’s Second Law

F=ma  
Where F=force, m=mass, a=acceleration  
#Used to calculate gravity in gravity()

Drag  
Fdrag= Cd \*ρ\*v2/2\*A  
Where Fdrag=force of drag, Cd=drag coefficient (determined by shape of parachute (assumed to be circular)), ρ=air density (dependent on altitude), v=velocity, A=area  
<https://www.grc.nasa.gov/www/k-12/airplane/drageq.html>

Terminal velocity  
Fnet=mg-(1/2)CρAv2=0  
(used to find terminal velocity)  
Where Fnet=force of terminal velocity, m=mass, g=gravity, C=coefficient of drag, ρ=air density, A=area, v=velocity  
<http://hyperphysics.phy-astr.gsu.edu/hbase/airfri2.html>

# Purpose

* Why we did this project
* Relevance to real world

# Limitations

Simulation accuracy.

The simulator can only simulate up to 43,000m. This is due to the equations used in the physics file, which can be fixed later on to simulate extremely high altitudes, even though the stratosphere ends at approximately 50,000m.

The simulation does not take into account time. The simulation will run at full speed instead f moving objects in relation to real time.

# Attribution of Work

Simulation code; configuration incorporation; PID algorithm and tuning; physics file; GUI file: Wilkie  
Research and theory; physics and equations; PID algorithm and tuning; simulation code: Maddie

# References

<http://www.instructables.com/id/Plotting-real-time-data-from-Arduino-using-Python-/>

<https://gist.github.com/electronut/d5e5f68c610821e311b0>

<http://playground.arduino.cc/Interfacing/Python>

<https://developers.google.com/gdata/articles/python_client_lib#introduction>

<http://www.instructables.com/id/Flight-Simulator-with-Arduino-and-Python/>

<http://pyode.sourceforge.net/> # possible physics engine

<http://stackoverflow.com/questions/10167329/change-the-position-of-the-origin-in-pygame-coordinate-system>

Pygame: <http://www.pygame.org/hifi.html>