ECE163: Payload Delivery

Matthew Bennett, Chris Massiello, Rembert Sison, Brandon Mee-Lee

May 2021

Contents

1 Introduction

In this project we hope to solve the problem of delivering packages to target locations and target vehicles. To do so we intend to build on top of the existing simulator. Our goals are to be able to generate a target location or vehicle, direct the aircraft to fly over said target and release a package that will land at the target location. We will gauge our success based on the plane's ability to: navigate to the target, release the payload, and the proximity of the package's

landing site to the target. Our goal for a MVP (Minimum Viable Product) is to be able to calculate a drop point based on a given target, have the plane fly to a predetermined altitude at the drop point, and have the package drop with no aerodynamics, meaning that we would simulate the package falling with no forces do to drag or wind effects. It would be as if it were falling in a vacuum. With this, the system would be able to serve as a model for dispersing packages to an area automatically with a reasonable amount of accuracy.

Once we are successful in implementing the MVP, the system will be expanded upon to more accurately model delivery in a realistic environment. We intend to model the package's descent with it's own aerodynamics model that will respond to the effects of drag. At this point we will be modeling with no wind. Following this, we will predict the drop point at a given wind speed. With that working we aim to be able to simulate this process visually in the same environment as the existing simulator. Finally we hope to apply this system to deliver to a moving target. Given a target vehicle on the ground moving along a given course we hope to be able to deliver a package in the path of the vehicle.

2 Overview

For our project, we need to make classes for the payload and target and also update some of the modules we made throughout the course. We need to adjust how we control the flight of the UAV if we have enough time we would also like to make some sort of graphical representation for our package delivery system.

We need to make a payload class and declare physical constants for the size and shape of the payload. The functions for the payload class are listed below.

- Constants.PayloadConstants()
- Payload.__init__()
- Payload.reset()
- Payload.CalculateDropPoint()
- Payload.DynamicsModel()
- Payload.AerodynamicsModel()
- Payload.Update()

Inside of Payload.AerodynamicsModel() we would also need these functions:

- gravityForces()
- parachuteForces()

- aeroForces()

We also need to make a target class. The functions for the target are listed below.

- Target.__init__()
- Target.reset()
- Target.setPosition()
- Target.getPosition()
- Target.setVelocity()
- Target.getVelocity()
- Target.Update()

We need a module to calculate the drop point then use that drop point to control where the aircraft needs to fly to. This module also needs to declare an instance of the target, payload, and aircraft.

- PayloadDeliveryControl.__init__()
- PayloadDeliveryControl.reset()
- PayloadDeliveryControl.UpdateDelivery()
- PayloadDeliveryControl.CalculateDropPoint()
- PayloadDeliveryControl.CalculateChi()
- PayloadDeliveryControl.Update()

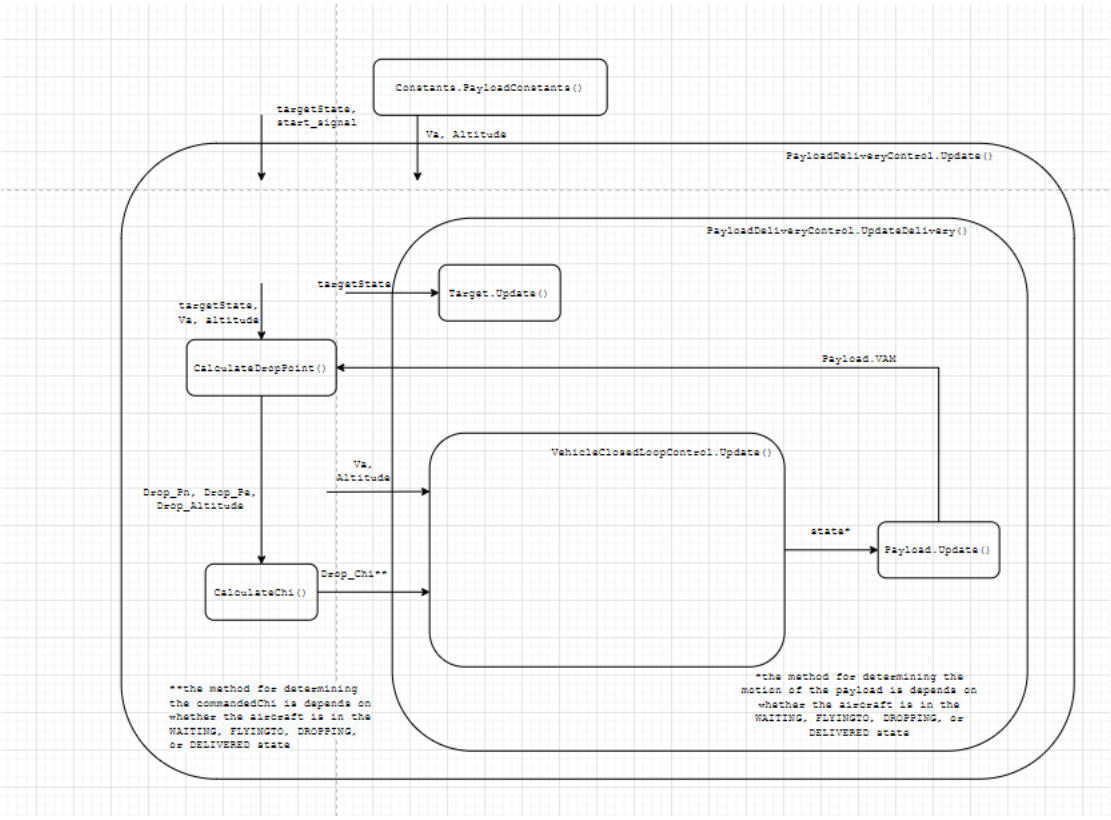

Figure 1: An information flow diagram for PayloadDeliveryControl

3 Module Details

3.1 Constants.PayloadConstants()

Constants.PayloadConstants will define the physical parameters for the payload. The payload will be a cube that will be small enough so that it could fit inside the model of the UAV.

3.2 Target.init(state) and Target.reset()

The target class will have a state passed in. From this state we will be using the Pn, Pe,u and v. With these variables the target will have a defined position and if the target is a moving vehicle it will have a velocity we can track.

3.3 Target.setPosition(Pn, Pe) and Target.getPosition()

These functions are wrapper functions that will simply set and get the position of the target. The target will not change in altitude so there will be no Pd input.

3.4 Target.setVelocity(u, v) and Target.getVelocity()

Simple wrapper functions for setting and getting the velocity of the target. The target will not change in altitude so w won't be an input. These functions will only be implemented if we achieve our stretch goal of delivering the payload to a moving target.

3.5 Target.Update(state)

Target update will only be needed in the case where we have a moving target. It will take the previous position of the target and move it forward the distance it would move in one timestep.

3.6 Payload.init(state, aerodynamics*) and Payload.reset()

Payload will have a state which keeps track of the position and velocity of the payload itself. If we are able to implement the payload to interact with the air and wind it will also have an aerodynamics model. If we were to add the aerodynamics model then we would have to change the calculation of the aerodynamic forces inside of it.

3.7 Payload.SetAerodynamicsModel()

Wrapper function to set the instance of AerodynamicsModel that is a parameter to the Payload class.

If we reach a position where we implement a parachute that opens mid free fall of the payload this function would also be used to change the aerodynamics of the payload from its pre parachute behavior to its behavior with the parachute.

3.8 Payload.Update()

This function will take the payloads current position and velocity and calculate the next velocity based on the acceleration due to gravity and then determine the position the payload would be in next based on that.

If we reach the a level with the aerodynamics acting on the payload then these effects would also be taken into account when determining the next position of the payload.

3.9 PayloadDeliveryControl.Init() and reset()

PayloadDeliveryControl() will be wrapped around the VehicleClosedLoopControl, generating control commands for altitude and course angle based on our desired drop location. PayloadDeliveryControl() will need curPayload Payload object, a curTarget Target object, and a VCLC instance for the plane itself.

3.10 PayloadDeliveryControl.UpdateDelivery(start,targetState)

UpdateDelivery will be the workhorse of the entire system. It will be used to call the all lower modules needed updates.
It will be passed in a targetState and an start signal. When passed a valid activation signal it will take the targetState and use that to generate a valid target using Target.setPosition(Pn, Pe) and Target.setVelocity(u, v) to create and if applicable move our target. It will also call all of the update functions we need, including Payload.Update(), VehicleClosedLoopControl.Update(), and Target.Update().

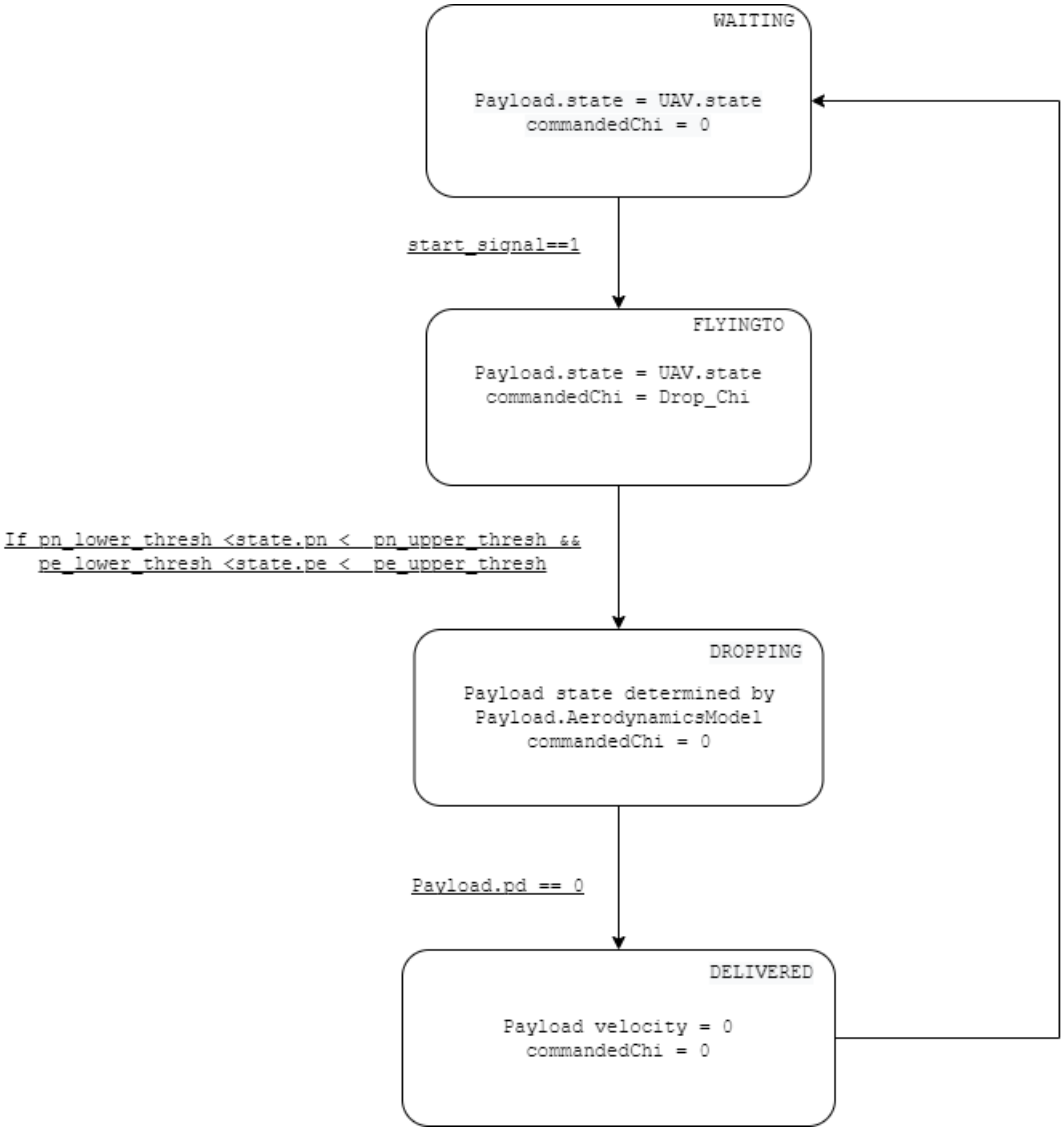

Figure 2: A state machine diagram for handling modes and transitions in PayloadDeliveryControl

The function will also maintain a state machine made up of a WAITING, FLYINGTO, DROPPING, and DELIVERED states. On startup of the simulator, the default state will be WAITING, as no delivery coordinates will have been passed through. Once the method DeliverToCoordinates() has been passed a valid activation signal the state machine will transition into the FLYINGTO state. In this state UpdateDelivery() will start calling CalculateDropPoint() based on the position of the target.The output of CalculateDropPoint() and the current position and course angle of the plane will be passed to CalculateChi(). The output of CalculateChi() will be our new commanded angle and use its output to determine the commanded course we want to pass to the system VehicleClosedLoopControl.Update(). When in the FLYINGTO state once the plane reaches the drop point, we'll "release" the payload. This is represented by us moving to the DROPPING state. In the dropping state the payload will now be Updated based on its own movement. Before the dropping state the payloads position will be the same as the plane. Once we reach the dropping state however the Payload.Update will now start calculating the payloads new position. In this state the plane will return to level flight at the default commanded course. Once the package hits the ground, in this case that will be represented by the payload reaching an altitude of 0, we will transition into the DELIVERED state. In the DELIVERED state we will get the position of the of the package and the position of the target and compare the two values to see how close the package was to its location. This data will then be stored in an array to be analyzed later. Following this the plane will transition back to the WAITING state.

3.11 PayloadDeliveryControl.CalculateDropPoint()

The CalculateDropPoint method will take in the airspeed, altitude, target position, and target velocity and calculate the point to where the UAV should release the payload. In the MVP this will be simple as we just take the distance the payload will travel laterally(motion in N and E) in the time it will take to fall. We will assume that the package initially has a velocity of 0m/s down when it starts accelerating. And it will have have the same velocity as the plane in the North and East directions. With this we can simply take the velocity of the payload * the time it will be falling to find its lateral movement. With that we can find the needed drop point. When we beign to factor in aerodynamics and wind we will have to determine the way that will affect the velocity of the package moving laterally. This will serve as an offset to the existing calculation and will need to be calculated throughtout the FLYINGTO state in case the wind changes mid flight.

3.12 PayloadDeliveryControl.CalculateChi()

The method CalculateChi will take the vehicle state and the output of CalculateDropPoint as inputs, and output the angle Chi that would result in a straight line being drawn between the current position and the target position. This is simply calculated by taking the tangent of ((dPn-state.Pn)/(dPe-state.Pe)). Where dPn and dPe are the Pn and Pe Values of the drop point.

3.13 PayloadDeliveryControl.Update()

Update function will be used by the simulator itself to call the UpdateDelivery() function with the start signal and target location passed in by the "user" in this case that user being the simulator.

4 Testing Strategy

To test the accuracy of our design our team will simultaneously utilize simulations and plots to allow observation of the UAV and package's behavior. Particularly, our design will have multiple design phases in which the following tests will ascertain the viability of the current phase's results.

4.1 Calculate drop point

We will simultaneously utilize simulations and plots observe the UAV and package's behavior.

- Trivial Case: Delivery airspeed of 0, no aerodynamics, delivery altitude of 100.
  - we would expect the delivery point to be 100m above the target location.
- Standard Case: Given a predetermined delivery airspeed of 25 m/s.
  - With no air resistance we would expect the payload's horizontal velocity(Va) to be maintained. With this, the drop distance can be calculated by:

    $$V_a \cdot \sqrt{\frac{2P_d}{g}}$$

    Therefore we would expect the drop point to be 112 m ahead of the payload's course.

4.2 Plane flies to predetermined altitude at drop point

As the simulation progresses, the UAV and payload's current position and velocity are plotted as they travel towards the drop point. The following cases are tested:

- Position/Velocity of UAV and payload should be equal during its flight towards the drop point.
- At the drop point, the airspeed of the UAV and payload should be equal to the predetermined delivery airspeed.
- Once the UAV is at the drop point, is it in a state of level flight?

4.3 Drop payload without aerodynamic model

Equaling the previous phase, the UAV and payload's current position and velocity are plotted as the payload descends towards the target location. The PayloadDeliveryControl class has an attribute that is a Payload class containing the payload's Pn, Pe, Pd, u, v, w. On each Update() call, the state is stepped forward, with a constant acceleration of g in the down direction. During this descent, the following cases are tested for:

- As the payload is "released" from the UAV, its position is expected to appropriately differ from the UAV's own position. If not, the package was unsuccessfully "released."
- Once the payload's Pd is equal to 0, its velocity in each direction should equal 0 to emulate touchdown. If not, the payload is incorrectly falling or moving in the East or North direction.
- For multiple iterations of the simulation, once the payload's Pd is equal to 0, its current position is plotted against the target's location to ascertain if the payload is within our 1 m success threshold.