The focus for class the next three classes will be requirements for the project. The requirements need to identify the elements we will work on throughout the semester. As such, they must address the competition rules, constraints of the existing vehicle and the trap. At the time of creating this document, I cannot yet directly give you the trap details (that I showed you in class – we will be able to look at it again during class) or CAD drawings for the vehicle. I am hoping you will have access to that information in the next few days.

I did show you a summary document about the UAV. You can access that Google Doc at: <https://docs.google.com/document/d/1kPdElo-byl_CzSKBHRLwS1xsHquRDY12y8hjxma86EQ/edit?usp=sharing>

There is also a detailed parts list, which contains the same information as the summary, but with more detail: <https://docs.google.com/spreadsheets/d/1a1KtV7vPQgCy3XI2gCwAEm253rJVnlgVdCtUNyuoxTA/edit?usp=sharing>

The competition rules are in flux, and there will be modifications to them over the next 24 hours, but we do have something to work with. I am attaching the current competition rules at the end of this document.

**PixHawk Information:** PixHawk is the autopilot system for the UAV. If you would like to get some general idea of what it is and how it works, the following are a few links:

General PixHawk Information: <https://store.3dr.com/products/3dr-pixhawk>

There is also a general website dedicated to additional information: <https://pixhawk.org/start>

On that website you can find a bunch of tutorials: <https://pixhawk.org/users/tutorials> and a user guide and developers guide: <http://px4.io/support/documentation/> as well as <http://dev.px4.io/starting-initial-config.html>.

**ROS Information:** Many of you are not familiar with the Robot Operating System (ROS), thus Anton put together a virtual machine that you can download and begin to become familiar with ROS. ROS is the primary middleware operating system, but there are also simulation tools associated with ROS called Stage (2-D) and Gazebo (3-D).

The VM image is quite large, but you can download it from: <https://vanderbilt.box.com/s/vhzvx4h0xolkutabodouwi371yc84xfs>.

There are instructions for importing this to VirtualBox: <https://www.virtualbox.org/manual/ch01.html#ovf>.

Virtual Machine login information:

Username: uav

Password: vandyuav

The VM archive should work with other virtualization applications, but Anton has not used others as much as VirtualBox, so he will be unable to help with those.

A native Ubuntu 14.04.3 (XUbuntu, KUbuntu, etc. are fine) install is preferred for performance reasons, but the VM will work for learning the basics of ROS. The VM has ROS Indigo installed, the most recent LTS release from May 2014. It was selected because it's an LTS. That being said, Jade has been out for almost 8 months now and is probably stable enough and has enough support that it can be used for the class. If you want a native install please see step one in the first tutorial section below and you can email Anton if you have any issues. It's simple apt-get so there shouldn’t be any issues. The tutorials should work with either the Indigo or Jade release.

<http://wiki.ros.org/ROS/Tutorials>

All the beginner level tutorials are recommended for students touching any code. This covers how to create and build packages in ROS, data visualization, pub/sub architecture, messages, client/service functionality, and debugging messages between nodes.

<http://wiki.ros.org/tf>

The TF package will be helpful for keeping track of the location of objects in the environment.

<http://wiki.ros.org/catkin/Tutorials>

Catkin is the build system.

<http://wiki.ros.org/turtlebot_stage/Tutorials/indigo/Bring%20up%20TurtleBot%20in%20stage>

<http://wiki.ros.org/turtlebot_stage/Tutorials/indigo/Customizing%20the%20Stage%20Simulator>

Stage tutorials to get started with a simulator, albeit a 2D one. Also good for seeing a larger system of ROS nodes and messages.

<http://gazebosim.org/tutorials>

Gazebo is a more appropriate simulator for UAVs with ROS because it is 3D. However, this will likely NOT work with the VM due to the virtualization performance hit. Students with more powerful laptops than mine (Macbook Pro with Core i5, 8 GB RAM, integrated graphics) may be able to run this in a VM.

Draft competition rules

CPS-VO2 (version: 2 Dec 2015)

# Summary:

We invite university students from to participate in the (project name) Challenge. The challenge will provide engineering students the opportunity to validate their analytic studies through a real-world vehicle design and verification experience.

Student teams will architect, design, fabricate, predict, and demonstrate the flying and maneuvering capabilities of an unmanned, electric radio controlled or autonomous vehicle that can best meet the specified mission profile. The goal is a balanced design possessing good demonstrated qualities with and without a payload, reliably pickup/dropoff of payload, practical and affordable manufacturing requirements, and high overall vehicle performance.

# Judging

# Competition Site

Some details of the competition site will need to be finalized in order to refine the missions and scoring.

# Schedule

## Framework schedule:

# Vehicle requirements

The vehicle is based off of the DJI Flame Wheel F450 airframe with the Pixhawk flight management unit (FMU) built by the University of Pennsylvania.

The Payload must be retrieved, transported, and dropped off as part of the competition.

GPS will be available during the final competition.

# Missions for final demonstration

## Mission Sequence (general)

The sequence of mission events is described below:

1. Mission-specific details are shared with the team
2. Team has a short time (~10 minutes?) to respond with updated artifacts and configuration request (if appropriate).
3. Mission is performed
4. Data acquired are used to score the mission.

Missions are decomposed into two basic types. *Vehicle* challenges perform tasks with the vehicle in the loop. *System* challenges provide answers to system questions, based on the parameters measured in Vehicle missions.

## Vehicle Challenge 1: Time prediction for payload pickup/transport/dropoff

**Overview**: Starting from any location L0, pick up a payload from a defined location L1, bring payload to the second location L2, and drop it off before going to location L3 (a recharge station) and land on the recharge station.

* Three attempts are allowed
* After each mission specification, team has 10 minutes to provide:
  + updated model (if desired)
  + a time bound during which they think that the mission can be completed.
* Score = T+M
  + The lowest score, calculated by the product of the
    - T is the time required (in seconds), made up of
      * T\_air time the vehicle is in the air
      * T\_ground time the vehicle is on the ground
    - M is the time bound (in seconds), estimates as T +/- M
    - T\_auto is any time spent in autonomous mode during pickup and dropoff
    - k is a multiplier to penalize sitting near the final waypoint until T

## Vehicle Challenge 2: Measurement of deviation from trajectory with payload

**Overview**: Waypoints (L0, L1…) provided, the vehicle (with payload already attached) takes off, flies to each of these waypoints, and lands at the last waypoint.

* Three attempts are allowed
* After each mission specification, team has 10 minutes to provide:
  + updated model (if desired)
  + an expected trajectory, based on the simulated model flying the same waypoint set.
* Score: min( Score 1, Score 2 )
  + Score 1: sum of squares error to the nearest point of the straight lines drawn between each waypoint (within a certain error is 0, e.g., less than 1m)
  + Score 2: sum of squares error to the nearest point of the expected trajectory provided by the Team’s simulation (within a certain error is 0, e.g., less than 1m).

## Vehicle Challenge 3: Measurement of timed deviation from trajectory with payload

**Overview**: Timed Waypoints (TL0, TL1…) provided, the vehicle (with payload already attached) takes off, flies to each of these waypoints and loiters (or lands) for a time centered around the required time, and arrives at the last waypoint at the required time.

* 1 attempt is allowed
* After mission specification, team has 10 minutes to provide:
  + updated model (if desired)
  + a model that explains the time range justification
  + an expected timed trajectory, based on the simulated model flying the same waypoint set, including time ranges during which they are guaranteed to be at each waypoint
* Score: min( Score 1, Score 2 )
  + Score 1: sum of squares error to the nearest point of the straight lines drawn between each waypoint (within a certain error is 0, e.g., less than 1m); the error is taken from the nearest point of the segment for which the time is valid
  + Score 2: sum of squares error to the nearest point of the expected trajectory provided by the Team’s simulation (within a certain error is 0, e.g., less than 1m); the error is taken from the nearest point of the trajectory for which time is valid.

## System Mission 1: Air-Traffic Controller

**Overview**: Team must provide a timed sequence of waypoints to move the vehicles to their final locations, using values from Vehicle Mission 3.

* 1 attempt (Note: scoring is done in simulation)
* Team receives an area description in which vehicles can fly (or not), a list of initial Vehicle locations, a list of final Vehicle locations, and the Vehicle Mission 3 parameters from each of the teams.
* Team provides
  + Timed waypoints sequence for each Vehicle
* Simulator
  + Executes the series of timed waypoints using (vehicle models, over approximations)
* Scoring
  + Minimum time to complete
  + Minimum time in the air for each vehicle
  + Subtract points if UAVs get within (distance 1) of one another; 0 points for the entire Mission if any UAVs get within (distance 2)

## System Mission 2: Strategic Deployment of Vehicles and Sensors

**Overview**: Team chooses how to allocate sensors in order to obtain reliable coverage for a disease with a specific incubation period

* 1 attempt is allowed (Note: mission is in simulation only)
* Team receives the description of an area for coverage, specified as an ordered set of Waypoints that describe a (potentially convex) polygon, and one unique waypoint as the pickup/dropoff location for mosquito traps, and models provided as part of Vehicle Mission 3.
* Team provides
  + locations for mosquito traps (in Waypoint format)
  + locations of recharge stations (in Waypoint format)
  + initial location of UAVs to refresh the mosquito traps (in Waypoint format)
  + plan to fly UAVs to each waypoint, avoiding other UAVs by design (not with sense-and-avoid)
* Simulator:
  + Uses the UAV models in Vehicle Mission 3
  + Executes the sequence of timed waypoints for each vehicle,
* Score (need to codify, still)
  + Takes into account number of UAVs needed
  + Uses time to pickup/dropoff based on the largest value in Vehicle Mission 1
  + Vehicle cannot run out of battery
  + Vehicle Team: receives points if their Model scores best