ROS Setup for UAV Collision Avoidance Simulation

J. Matthew Holt

May 20, 2011

1 Introduction

Unmanned Aerial Vehicle (UAV) research is an area of great interest, especially for data collection. Usually, when trying to gather data using UAVs, many UAVs may be used in order to collect data faster or provide redundancy. In these scenarios, some collision avoidance needs to be implemented in order to prevent the multiple UAVs from colliding with each other mid-air. However, it becomes difficult to test collision avoidance with using the actual planes (which may lead to collisions during the testing of the avoidance algorithm). In order to prevent this, a simulation is needed in order to test the collision avoidance algorithms and provide methods for evaluating whether A) the algorithm is working and B) a simulated collision has occurred. For this simulation to be a success, the algorithm and control mechanisms shouldnt be able to tell the difference between simulated planes and real planes. In order for this to work, we propose a system using the Robot Operating System (ROS) so that the simulation and real planes can be run independently or even concurrently. This will allow for the safe testing of avoidance algorithms before being used in the field.

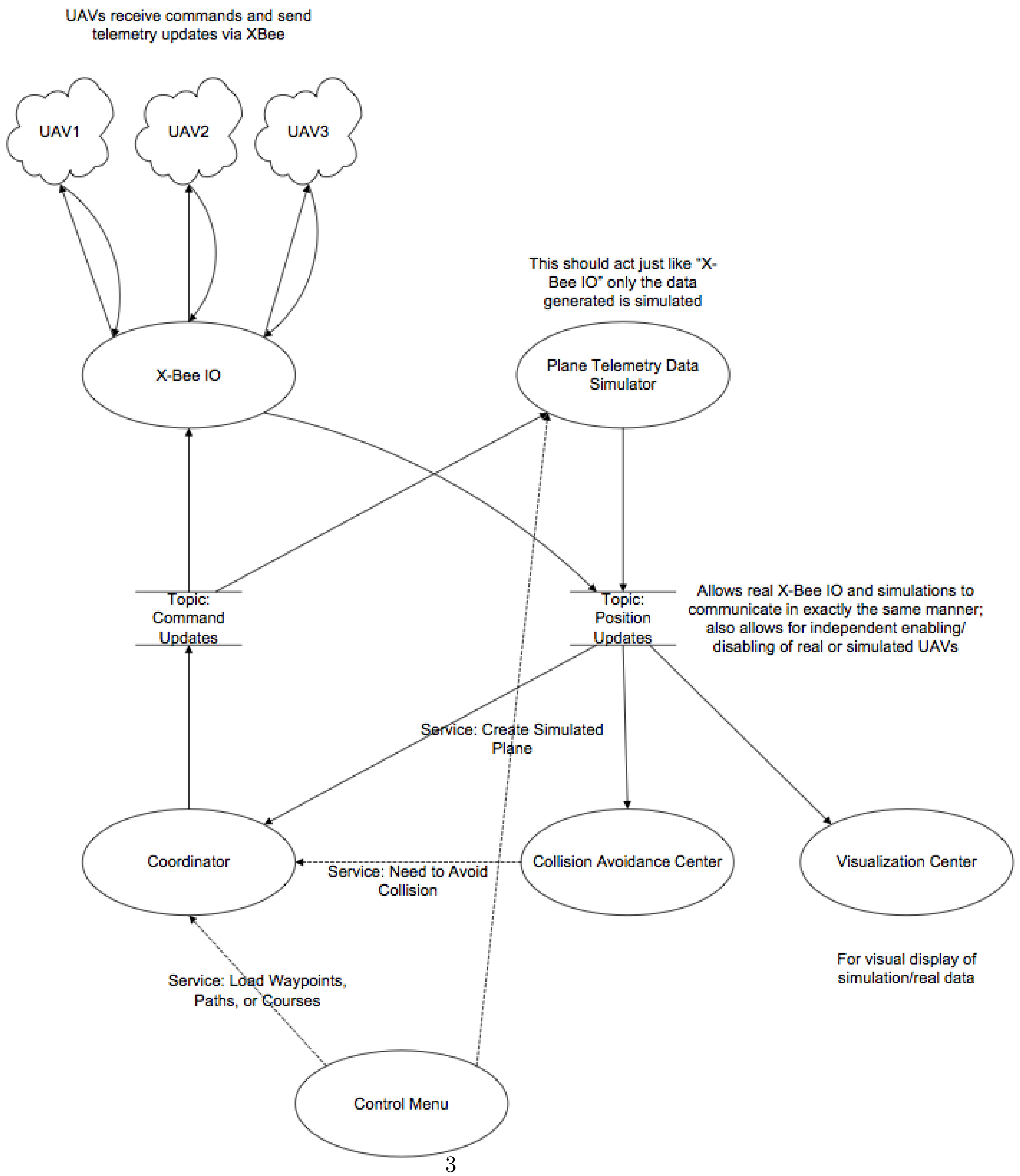
2 System Overview

The system as a whole can be divided into five major categories and ROS nodes, each of which will be described in this document in greater detail:

1. Coordinator - Responsible for storing plane telemetry data and sending new commands to the UAVs
2. Collision Avoidance Center - Responsible for monitoring known UAVs and rerouting to avoid collisions
3. XBee IO - Responsible for transmitting and receiving data via Arduino to the ArduPilot

4. Plane Telemetry Simulator - Responsible for receiving commands and the simulation of any UAVs flight data
5. Visualization - Responsible for receiving telemetry updates and displaying the data on screen
6. Control Menu - Responsible for handling user input to manage the system including simulated plane creation and loading paths/waypoints into the system

3 Current System Diagram - 5/20/11


UAVs receive commands and send telemetry updates via XBee
UAV1
UAV2
UAV3
X-Bee IO
This should act just like "X-Bee IO" only the data generated is simulated
Plane Telemetry Data Simulator
Topic: Command Updates
Topic: Position Updates
Allows real X-Bee IO and simulations to communicate in exactly the same manner; also allows for independent enabling/disabling of real or simulated UAVs
Service: Create Simulated Plane
Coordinator
Service: Need to Avoid Collision
Collision Avoidance Center
Visualization Center
For visual display of simulation/real data
Service: Load Waypoints, Paths, or Courses
Control Menu

4 Communication Standards:

4.1 Position Updates

4.1.1 Description:

In order for the simulator to be as real as possible, the communication of position updates must be exactly like the ones received from real UAVs. For this purpose, a standard position update message format was created. The only two components that will be publishing to this topic are the real X-Bee IO and the Plane Telemetry Data Simulator. However, many components read from this topic and more can be added if the project requires it or future additions need it. This allows for expansion of the project with relative ease.

4.1.2 Message Format:

Header - standard ROS header for timestamps and other useful information
Plane ID - integer, associated with a given plane
Latitude - float, latitude coordinates of UAV
Longitude - float, longitude coordinates of UAV
Altitude - float, altitude coordinates of UAV
Next latitude - float, latitude coordinates of destination of UAV
Next longitude - float, longitude coordinates of destination of UAV
Next altitude - float, altitude coordinates of destination of UAV
Ground speed - float, speed relative to ground (meters/second)
Target bearing - float, bearing of the UAV
Current Waypoint - integer, index of the current waypoint
Waypoint Distance - float, distance to current waypoint

4.1.3 Publishers:

X-Bee IO, Plane Telemetry Data Simulator

4.1.4 Subscribers:

Coordinator, Collision Avoidance Center, Visualization Center

4.2 Command Updates

4.2.1 Description:

The commands being sent to UAVs also need to be standardized so that the real IO and simulated UAVs can be used interchangeably. To solve this, the command update message described below was created. Presently, only the coordinator will be set up to write to this topic because it should be the only one sending commands to any UAV (real or simulated).

4.2.2 Message Format:

Header - standard ROS header for timestamps and other useful information
Plane ID - integer, associated with a given plane
Go to latitude - float, latitude coordinates of command
Go to longitude - float, longitude coordinates of command
Go to altitude - float, altitude coordinates of command

4.2.3 Publishers:

Coordinator

4.2.4 Subscribers:

X-Bee IO, Plane Telemetry Data Simulator

5 ROS Node Descriptions

5.1 X-bee IO

5.1.1 Description:

This node is primarily responsible for acting as a relay between the ROS core of the system and the actual UAVs. Presently, there is JAVA code that would allow communication from a base station to these UAVs. This node will integrate with that JAVA code to provide a ROS plug-in for communication with the real UAVs. The major functionality will be translating internal ROS data into external X-Bee packets to be sent to the UAVs and vice-versa. Note that this means we will be changing from the internal normal representation of latitude and longitude to the ArduPilot format for latitude and longitude. Only computations specifically related to this translation and sending a packet should be performed in this node.

5.1.2 Publishes to:

Position Updates

5.1.3 Subscribes to:

Command Updates

5.1.4 Services accessed:

Request Plane ID - Coordinator

5.1.5 Other I/O:

X-Bee Communication with ArduPilots

5.2 Plane Telemetry Data Simulator

5.2.1 Description:

This node is responsible for calculating the data generated by simulated UAVs. In order for the simulation to be as real as possible, it will respond to command updates and generate position updates just like the real UAVs. This node will have fairly extensive math and physics based components due to the nature of the data being generated. Data (or acceptable simulation performance criteria) will need to be gathered on UAV update frequency, turn radius, flight control, etc. in order to get a reasonable approximation of how the real UAVs operate. At present time, one instance of the simulator process will be responsible for handling all simulated UAVs.

5.2.2 Publishes to:

Position Updates

5.2.3 Subscribes to:

Command Updates

5.2.4 Services provided:

Create simulated plane - This service will be called by the control menu whenever a user wishes to instantiate a simulated UAV. It will start the position updates and simulation of flight automatically.

5.2.5 Services accessed:

Request Plane ID - Coordinator

5.2.6 Room for future improvements:

Better simulation calculations
Handling environmental conditions

5.3 Coordinator

5.3.1 Description:

Contained within the coordinator will be the received position updates of any UAV along with the ability to store a planned path for any given UAV. Presently, this node is also the only node that will publish commands to be executed by real or simulated planes. Because of this, any collision avoidance or commands generated by other processes will have to go through the coordinator. While ROS does not create this limitation, this design decision forces any commands to be recorded and handled appropriately by the coordinator and allows the coordinator to put the UAVs back on track after a collision is successfully avoided.

The path planning works by having two separate queues: normal path and collision avoidance path. If a waypoint is in the collision avoidance path queue, it has priority and is what will be transmitted to the UAV. However, if the avoidance queue is empty, the normal path is what will be followed. This setup allows the coordinator to prioritize the safety of the UAVs.

5.3.2 Publishes to:

Command Updates

5.3.3 Subscribes to:

Position Updates

5.3.4 Services provided:

Go To Waypoint - This is a general purpose routing function. If a user wishes to send a plane to a particular destination, this service can be used. Also, this service will be called by the Collision Avoidance Center whenever it detects a potential collision. It will provide information to the coordinator on what preventative actions need to be taken and the coordinator will send the appropriate commands to avoid a collision.

Request Plane ID - This service will return a plane ID for a plane (real or simulated) to use throughout the duration of the test.

5.3.5 Room for future improvements:

Collision avoidance service may need modification based on algorithms

5.4 Collision Avoidance Center

5.4.1 Description:

The collision avoidance center is a computation node responsible for monitoring the UAVs and generating avoidance maneuvers. Presently, its set up to monitor any incoming data from UAVs and base its calculation solely on this information. In the event of collision detection, the avoidance center will send corrective action to the coordinator through the avoid collision service provided by the coordinator. During design of this node, we will be providing a base skeleton for all communication for this node. However, we will be leaving the implementation of an avoidance algorithm to the summer students.

5.4.2 Subscribes to:

Position Updates

5.4.3 Services accessed:

Go To Waypoint - Coordinator

5.4.4 Room for future improvements:

As of now, no algorithm is implemented in this node. Future development will take different algorithms and special ROS improvements to accommodate those algorithms in order to create the collision avoidance algorithms.

5.5 Visualization Center

5.5.1 Description:

The visualization center will be the ROS node that handles displaying position data in a more visual environment. Presently, the plan is to use a pre-built ROS program called rviz for 2D plotting of UAV location and direction. This will allow for a user to visually monitor both real and simulated planes to determine the effective of collision avoidance and flight plans. We want to be able to watch the UAVs as they fly and view the path taken at the end.

5.5.2 Subscribes to:

Position Updates

5.5.3 Other I/O:

Visual Output of 2D UAV Path

5.5.4 Room for future improvements:

3D interface
Point and click path planning

5.6 Control Menu

5.6.1 Description:

The control menu is intended to be the command line menu-based interface for this system. For now, it will be capable of allowing the user to create simulated planes, send planes to a waypoint, load a path for a plane, and load a course for multiple planes. Eventually, this UI will probably be replaced.

Plane creation and loading a single waypoint are the two most basic components of this interface. Plane creation only requires a starting location and bearing. This simulated plane wont do anything until a waypoint is received, either from user input or a path being loaded. The single waypoint simply requires a UAV ID number and a waypoint to be input in order to start the UAV movement.

Loading a path or course will involve a special file format in order to layout what normal path should be loaded into a single plane or multiple planes (for a course). This format is yet to be determined but should not contain much beyond plane IDs and a series of waypoints.

5.6.2 Services Accessed:

Create Simulated Plane - Simulator Go To Waypoint - Coordinator

5.6.3 Other I/O:

User Inputs