Timothy Barao

Jonathan Clow

Eric Mustapick

COP5570: Concurrent, Parallel, and Distributed Programming

Dr. Zhenhai Duan

Term Project Final Report

December, 8, 2017

**Wingman: An ArduPilot Extension**

Wingman is a messaging extension to ArduPilot, an open-source drone control system for Arduino. Using the existing ArduPilot code base, the Wingman system enables users to control squadrons or swarms of connected drones through a single primary drone without significant additional hardware.

**Background**

With the rise of unmanned aerial vehicles, or UAV, and commercially available UAVs or drones, consumer and civilian use and interest in drone technology has seen prominence in recent years. From hobbyist applications to commercial enterprises, applications of UAVs have become more commonplace with examples including racing leagues and choreographed shows at Disney.

Building into the commercial and hobbyist spirit is ArduPilot. ArduPilot is a Linux-based open-source software suite that enables users to compile binaries for a given set of hardware and sensors and install said binaries on

**Proposal**

Initial proposal for the project was to develop a networking system for ArduPilot to allow a master drone to command a servant drone. In doing so, the user would send commands and updated flight coordinates to the primary drone, which would then communicate these received commands to the servant drone that would accept the commands and adjust course as appropriate.

Based off initial conception, a user operating a ground control station would send commands and waypoint updates to a designated master drone through existing systems. This drone would then communicate the updated commands and waypoints to other connected servant drones using prior established connections. Servant drones would then act accordingly for the given command. Communications between drones would be established in advance through user control and designation.

Given this conception, the project as proposed entailed an in depth investigation into ArduPilot command and control systems, including communication and networking between drones and the user operated ground station, processes for sending command messages between drones and the ground stations, message details, and identifying code points to extend the existing code base to send and receive commands between instances of drone binaries in operation.

**Research**

*“I spent a whole summer working on this and I still didn’t understand it.” – Dr. Zhenhai Duan*

**Development**

*“What these guys are doing is hard.” ~ Dr. Zhenhai Duan*

**Challenges**

**Limitations**

**Results**

**Appendix**

**Installation and Compilation of ArduPilot (Linux)**

**NOTE:** We have attempted to install and compile this on multiple systems via multiple means. The most successful approach has been to either use a Linux system or use a Windows 10 system through Bash for the Windows Subsystem for Linux (Ubuntu through the Windows Store).

* Clone the ArduPilot project from the Github repository
  + git clone httpsL://github.com/ArduPilot/ardupilot.git
* Install the system prerequisites and update the submodules
  + git submodule update –recursive –init
  + ./ardupilot/Tools/scripts/install-prereqs-ubuntu.sh
* Ensure the system dependencies are installed
  + g++-4.9
    - sudo apt-add-repository ppa:ubuntu-toolchain-r/test
    - sudo apt-get update
    - sudo apt install g++-4.9
  + cmake
    - sudo apt-add repository ppa:george-edison55/cmake-3.x
    - sudo apt-get update
    - sudo apt install cmake
  + gcc-arm-embedded
    - sudo apt-add-repository ppa:team-gcc-arm-embedded/ppa
    - sudo apt-get update
    - sudo apt install gcc-arm-embedded
  + sudo apt-get update
  + sudo apt-get upgrade
* Use the waf build system to compile a given vehicle
  + Note: while waf can build for multiple types of Arduino based boards (see ./waf list\_boards), we will be compiling the ArduCopter project with a given board of SitL (Software in the Loop) for simulation purposes
  + From the base Ardupilot directory
    - ./waf configure –board sitl
    - ./waf copter
* Running the compiled copter binary in simulation
  + From the base ArduPilot directory
    - cd ./build/sitl/bin
    - ./arducopter –model QUAD –home 30.445969,-85.299795,32,0
      * This starts an ArduCopter simulated instance in the parking lot behind the Love building at an estimated altitude for ground level. (Estimated altitude is for Flight Gear home point – anything less than this value [32] will display as underground)
  + **NOTE:** As detailed in the Limitations section, compilation of any project is difficult at worst and flaky at best. If compilation fails for any reason, retry the compilation. We have found that typically a subsequent compilation will succeed at best and at least progress further through the binaries at worst, resulting in the need for an additional compilation.
    - **Other potential fixes:**
      * sudo apt-get install python
      * sudo apt-get install genromfs unzip zip python-empy
      * pip install future lxml
* Installing Mission Planner (Windows)
  + Download the software: <http://firmware.ardupilot.org/Tools/MissionPlanner/MissionPlanner-latest.msi>
  + Run the installation utility
  + Run Mission Planner
* Installing APM Planner 2 (Linux)
  + Download the software: <http://firmware.ardupilot.org/Tools/APMPlanner>
  + Install the APM Planner
    - sudo dpkg -I apm\_planner\*.deb
      * If install fails, install the prerequisites: sudo apt-get -f install
      * Retry the installation
  + Run APM Planner 2
* Connecting the APM Planner instance to the Copter instance
  + Go to File > Advanced Mode
  + Go to Communication > Add Link > TCP
  + Use these details
    - Host: 127.0.0.1
    - TCP Port: 5760
    - Connect
* Connecting MAVProxy to the simulation
  + Install MAVProxy
    - sudo apt-get install python-dev pythonopen-cv python-wxgtk3.0 python-pip python-matplotlib python-pygame python-lxml
    - pip install MAVProxy
    - Export the MAVProxy path to the system path
      * Echo “export PATH=$PATH:$HOME/.local/bin” >> ~/.bashrc
    - Change the user permission to allow dialot
      * sudo adduser <username> dialout
  + Connect the APM Planner instance to the ArduCopter instance
    - NOTE: This is required to open additional ports to the drone
  + Connect the MAVProxy instance to the drone
    - mavproxy.py -master=tcp:127.0.0.1:5760 –map –console
* Running multiple instances of the compiled drone
  + Run the compiled drone with an additional -I flag
    - ./arducopter –model QUAD –home 30.445969,-85.299795,32,0 -I 0
      * This starts the drone on port 5760
    - ./arducopter –model QUAD –home 30.445969,-85.299795,32,0 -I 1
      * This starts the drone on port 5770
  + NOTE: Any number of instances are allowed, with each incrementing the base port by 10 \* <instance number>

**Adding Wingman Extension to Existing Codebase**

* Untar the project code
* Drag the new code to overwrite existing code in the ArduPilot project
* Recompile the code with waf
  + ./waf copter

**Running the Wingman Protocol**

* Start two instances of copter drones as detailed above
* Start two MAVProxy instances connected to both drones as detailed above
  + mavproxy.py –master=tcp:127.0.0.1:5772 –map –console –load-module connect
    - This loads the custom MAVProxy module “connect” which we designed to send the custom MAVLink message via MAVProxy
* From MAVProxy on one drone, issue command
  + connect <port of second drone>
    - i.e. connect 5772
* Issue commands to the initial MAVProxy instance as normal
  + i.e.
    - mode guided
    - arm throttle
    - takeoff 40
      * Note: this command must be issued within 15 seconds of arming the throttle or else the drone’s engines will disarm