Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Incorporating Commercial Autopilots into DRDC's Middleware for Robotics (MIRO)

S. Monckton
DRDC – Suffield

R. Desgagnés
AEREX Ltd.

E. Gagnon
DRDC – Valcartier

M. Lauzon
DRDC – Valcartier

Defence R&D Canada – Suffield

Defence R&D Canada

Technical Memorandum

DRDC Suffield TM 2010-178

December 2010

Canadä

Incorporating Commercial Autopilots into DRDC's Middleware for Robotics (MIRO)

S. Monckton
DRDC – Suffield

R. Desgagnés
AEREX Ltd.

E. Gagnon
DRDC – Valcartier

M. Lauzon
DRDC – Valcartier

Defence R&D Canada – Suffield

Technical Memorandum

DRDC Suffield TM 2010-178

December 2010

Principal Author

Approved by

D. Hanna
Head/Autonomous Intelligent Systems Section

Approved for release by

Dr Paul A. DAgostino
Chairman/Document Review Panel

Abstract

Defence R&D Canada (DRDC) is extending its considerable experience in teleoperated air, land and seaborne systems to the development of autonomous systems for the Canadian Forces. Not surprisingly commercial organizations have developed a variety of unique and proprietary autopilots for unmanned aircraft vehicle control. Typically each provider claims multi-vehicle control as part of their product capabilities, but the lack of common standards between manufacturers makes such capabilities functionally of little value. Though some emerging standards (such as JAUS) show promise, DRDC has recognized that proprietary controllers are unavoidable and have chosen to incorporate such systems into a larger multi-vehicle system based on CORBA, an open networking standard. This report describes the integration of a GCS/autopilot system into the MIRO environment.

Résumé

Recherche et développement pour la défense Canada (RDDC) désire accroître sa vaste expérience dans le domaine des systèmes aériens, terrestres et maritimes téléguidés en développant des systèmes autonomes pour les forces canadiennes. Bien entendu, des organismes commerciaux ont développé une variété d'engins autopilotés uniques et exclusifs servant à commander des véhicules aériens sans pilote. De manière générale, chaque fournisseur déclare que leur produit peut contrôler plusieurs engins en même temps, mais le manque de normes communes pour les fabricants fait en sorte que ces produits ne valent pas grand-chose sur le plan fonctionnel. Bien que certaines normes émergentes (comme les JAUS) semblent prometteuses, RDDC reconnaît que les contrôleurs privés sont indispensables, et a donc choisi d'intégrer ces systèmes au système multi-engins basé sur CORBA, une norme ouverte sur les réseaux. Le présent rapport décrit l'intégration du SCS/autopilote dans l'environnement MIRO.

This page intentionally left blank.

Executive summary

Incorporating Commercial Autopilots into DRDC's Middleware for Robotics (MIRO)

S. Monckton, R. Desgagnés, E. Gagnon, M. Lauzon; DRDC Suffield TM 2010-178; Defence R&D Canada – Suffield; December 2010.

Background: Unmanned Air vehicles have a long history of teleoperated control and limited waypoint following capabilities exploiting both GPS and INS sensing. Most autopilot providers use proprietary vehicle command interfaces and multivehicle control protocols. Though STANAG 4586 represents an existing industry standard multivehicle control interface for ground stations, few providers adhere to nonproprietary standards for network-enabled control. This report documents the integration of two autopilots, the Cloudcap Piccolo Plus and the Rotomotion autopilot into a single multi-vehicle control network through the use of DRDC's Middleware for Robotics or MIRO. Given the number and variety of air vehicles based on similar architectures, incorporating these devices into a common framework could provide a foundation to expand the capabilities of DRDC's control network.

Principle Results: The Cloudcap and Rotomotion autopilots embody two basic and formative UAS communication techniques: serial RF communications and wireless UDP communications. As a more recent standard based on 802.11 wireless networking, Rotomotion's wireless UDP solution is less mature than CloudCap's Piccolo autopilot, based on traditional serial wireless communication. With a longer commercial history, Cloudcap's ground control station is more mature than Rotomotion's, yet both fail to provide a network enabled nonproprietary multivehicle control comparable to DRDC's MIRO framework. Similarly, both systems use different aircraft dynamic simulators to train and visualize UAS operations. With more commercial history, the Piccolo API is both more capable and complex than the Rotomotion's. Despite this maturity, Rotomotion's nonproprietary networking approach makes vehicle communications relatively clear, portable, and extensible. As a result, the Rotomotion MIRO driver has no dependence on external code, using a proven publish-subscribe model founded on direct socket communication with the autopilot. In contrast, the resulting Cloudcap driver leverages proprietary libraries to establish a socket connection to the GCS and not the autopilot. While any driver must be updated with any changes to an API, this work suggests that MIRO's direct connection used for the Rotomotion driver remains a more difficult, but more portable cross platform solution.

Significance of Results: For the first time at DRDC, multiple independent commercial autopilots have been added to a single network-enabled open robot architecture. While successful in exploring both the likely avenues and problems of integrating commercial systems designed to proprietary standards, the work reveals weaknesses of proprietary approaches and identifies problems with the current MIRO infrastructure that limit both the simulation and visualization of multiple air and ground vehicles. Further, this work suggests the adoption of a generic autopilot service that glosses over the substantial structural differences between autopilot memory, state, and communication protocols.

Future Plans: This work clearly points to the necessity for a standard model of an idealized UAV mission control system that, in turn, can be mapped through appropriate drivers onto a specific system's GCS and/or autopilot. Further, MIRO's existing simulation infrastructure, Gazebo, must be federated with simulators of other dynamic environments, such as sea and air. While MIRO provides many powerful capabilities missing from commercial autopilot solutions, incorporating third party autopilots into a CORBA infrastructure is clearly only the first step to a genuinely seemless multivehicle simulation, visualization, and control environment.

Sommaire

Incorporating Commercial Autopilots into DRDC’s Middleware for Robotics (MIRO)

S. Monckton, R. Desgagnés, E. Gagnon, M. Lauzon ; DRDC Suffield TM 2010-178 ; Défense Recherche et Développment Canada - Suffield ; décembre 2010.

Contexte : Les véhicules aériens sans pilote possèdent depuis longtemps une commande téléguidée et des capacités de poursuite d’un point de cheminement limité tirant profit du GPS et du INS. La plupart des fournisseurs d’autopilotes utilisent des interfaces de commande privées et des protocoles de commande multi-engins. Bien que la norme STANAG 4586 représente une norme de l’industrie relative à l’interface de commande multi-engins pour les stations au sol, très peu de fournisseurs adhèrent aux normes du domaine public pour les commandes facilitées par réseau. Le présent rapport documente l’intégration de deux (2) autopilotes, le Cloudcap Piccolo Plus et le Rotomotion, en un seul réseau de commande multi-engins via Middleware for Robotics ou MIRO de RDDC. Étant donné le nombre et la variété de véhicules aériens basés sur des architectures semblables, l’intégration de ces deux dispositifs en un seul cadre de travail pourrait constituer une bonne base qui permettrait d’élargir les capacités du réseau de commande de RDDC.

Résultats : Les autopilotes Cloudcap et Rotomotion incarnent deux techniques de communication pour UAS : les communications RF en série et les communications sans fil PNU. Le système de communications sans fil PNU de Rotomotion constitue une solution plus récente que l’autopilote CloudCap, lequel est basé sur une communication en série classique. Sur le marché depuis longtemps, la station de commande au sol Cloudcap est plus évoluée que celle de Rotomotion, mais les deux stations sont incapables de fournir une commande multi-engins du domaine public facilitée par réseau comparable au cadre de travail MIRO de RDDC. De même, les deux systèmes utilisent des simulateurs d’aéronefs dynamiques différents pour l’entraînement et la visualisation des opérations UAS. Le Piccolo API, fort d’une longue présence sur le marché, possède une capacité supérieure et plus complexe que celle de la station de Rotomotion. Malgré son manque d’expérience sur le marché, l’approche de réseautage du domaine public de Rotomotion fournit un système de communication avec le véhicule relativement plus clair, portable et extensible que d’autres. Ainsi, le pilote MIRO de Rotomotion ne dépend nullement de code externe et utilise un modèle publication-abonnement éprouvé qui se base sur des communications directes à raccordement avec l’autopilote. À l’opposé, le pilote CloudCap tire profit des bibliothèques privées pour établir une connexion à contact avec le PCS et non avec l’autopilote. Bien que tous les pilotes doivent être mis à jour lorsque l’API est modifié, les présents travaux suggèrent que la connexion directe du MIRO utilisée par le pilote Rotomotion est une interplateforme beaucoup plus portable bien que plus difficile.

Importance des résultats : Pour la première fois à RDDC, plusieurs autopilotes commerciaux indépendants ont été ajoutés à une seule architecture robotique ouverte facilitée par réseau. Bien que RDDC ait connu du succès dans l'exploration des problèmes d'intégration les plus susceptibles de survenir avec les systèmes commerciaux conçus selon les normes privées, les travaux ont permis d'exposer les faiblesses des approches privées, et d'identifier les problèmes avec l'infrastructure MIRO actuelle qui limitent la simulation et la visualisation de plusieurs engins aériens et terrestres. De plus, les travaux recommandent l'adoption d'un service d'autopilote générique qui fait abstraction des différences structurelles substantielles entre la mémoire, l'état et les protocoles de communication de l'autopilote.

Recherches futures : Les présents travaux démontrent clairement la nécessité d'un modèle standard de système de contrôle de mission UAV idéal qui pourrait, en retour, être rattaché, via divers pilotes adéquats, à un PCS précis et/ou à un autopilote. De plus, l'infrastructure de simulation actuelle du MIRO, Gazebo, doit être fédérée avec les simulateurs d'autres environnements dynamiques, notamment en mer et dans les airs. Bien que MIRO offre de nombreuses capacités puissantes que d'autres solutions sur le marché n'offrent pas, incorporer des autopilotes de tierce partie à l'infrastructure CORBA constitue la première étape pour obtenir un environnement harmonieux de simulation, de visualisation et de commande multi-engins.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1 Introduction

With a long history of teleoperation research, Defence R&D Canada (DRDC) Autonomous Intelligent Systems Section (AISS) has embarked on autonomous systems development projects for the Canadian Forces. Unmanned Air vehicles have a long history of teleoperated control and, increasingly, limited waypoint following capabilities that exploit GPS and INS sensing. Though capable, most companies that provide such limited autonomy autopilots have extended their basic architecture to include proprietary vehicle command interfaces and multivehicle control protocols. Without an industry standard that imposes open interfaces onto these systems, consistent vehicle and multivehicle control irrespective of manufacturer remains elusive. This report documents the integration of disparate platforms into a single multi-vehicle control network through the use of DRDC's Middleware for Robotics or MIRO. Given the number and variety of air, land, and marine autopilots and the profusion of interfaces, the ability to incorporate these devices into the more sophisticated MIRO framework will greatly expand the scope of MIRO control.

This document describes the marriage of network enabled layers with proprietary or commercial autopilots, the necessary techniques and the possible benefits arising from this union. DRDC Suffield and DRDC Valcartier simultaneously explored strategies for mating network oriented upper level controllers to existing commercial controllers. Both centres used MIRO as the common network communications framework, DRDC Valcartier incorporated the nascent UNIX based Rotomotion helicopter Autopilot into MIRO using custom network drivers while DRDC Suffield developed MIRO control for the popular but more traditional Cloudcap Fixed Wing Autopilot using a blend of custom and commercial drivers.

1.1 Objective

The objective of this study is to explore and compare practical methods required to incorporate computationally proven, dedicated commercial vehicle microcontrollers into the evolving model of network enabled battlefield unmanned systems. By identifying alternative architectures, building, running and examining the resulting software, a qualitative assessment can be established on the relative merits and appropriate conditions of use for each method.

1.2 Background

With the rise of commercial Unmanned Air Vehicles (UAVs) and the similar commercialization of Unmanned Ground Vehicle (UGV component, developers of autonomous systems must decide whether to support or supplant commercial autopilot or vehicle control hardware. In [2], DRDC reviewed an internal architecture ANCAEUS to determine whether it constituted a 'foundation' for autonomy. The report concluded that though the hardware and software were reliable and, in many respects, visionary for their time, they would be unable to support truly distributed network enabled operations. Later reviews identified evolving frameworks and toolkits that had these capabilities [4].

As part of the Technical Investment Fund project “A unified approach for the Command and Control of UxV teams”, DRDC Suffield procured Rotomotion helicopters for both DRDC Suffield and DRDC Valcartier to augment DRDC Suffields fleet of ACR Silverfox Fixed Wing UAVs and Raptor UGVs.

In 2005, the ALS project [6] set a philosophical precedent of network based control coupled to simplified low level control.The Raptor UGVs, used extensively for the ALS project were controlled through a network of processes communicating through MIRO software to various sensing and high level control devices. Final low level control came from a custom built UGV MPC555 control module developed by XJ designs of Ottawa. This project demonstrated the utility and flexibility of such network control in the context of a complex sensing for a complex environment on a custom vehicle.

For all its success, the ALS project revealed some limits to the approach. Low level custom controllers are unique to the UGV domain – a commercial arena still in its early stages. This gives developers both the luxury and responsiblity of developing even the most basic route following logic. The more mature UAV market has largely dispensed with custom control, specializing into airframe, powerplant, and control manufacturers. Airframes can be quickly and readily mated to a variety of possible autopilots to produce useful capabilities in short order. Significantly, these autopilots encapsulate many of the problems encountered in the ALS project such as vehicle monitoring, basic route following, and GPS-IMU localization. If the future of UGV component markets proves to be similar to UAVs, how will the MIRO netcentric need to adapt? If UAV sensing and control complexity grow to match mission complexity, low level controllers and netcentric high level control networks will surely meet as UAVs descend lower in the Battlespace [7] or outside of guaranteed communications.

1.3 Middleware for Robotics (MIRO)

Robot systems integration has evolved from small hand built programs running on specialized computers to large software systems distributed over many processors. The difficulty of designing and maintaining flexible communications between multiple processes, each devoted to sensing, modeling, planning, and acting (or SMPA) drove the development or adoption of dedicated multiprocess control and communications software or middleware. In 2003, DRDC Suffield examined the middleware market and the needs of autonomous systems to assemble a toolkit of useful technologies. The Middleware for Robotics was selected as a provisional middleware standard for DRDC Suffield’s Autonomous Land Systems project demonstrated in 2005 [4]. As depicted in figure 1, MIRO serves to simplify multiprocess intercommunication common to large or distributed unmanned systems by using the Common Object Request Broker Architecture (CORBA) standard developed in the late 1990s. Embodying the wide scope of multinode networked communications and a decade of industrial development, CORBA can be difficult to grasp and complex to manage. Thus MIRO’s singular strength comes from forming CORBA into a set of simpler robot-specific capabilities with powerful software frameworks. Originally designed to support soccer robots at the University of Ulm, MIRO has become a powerful open source project that leverages popular collaborative efforts such as the more modest Player/Stage/Gazebo and Carmen

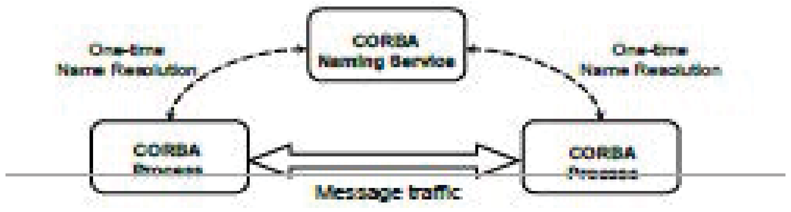


Figure 1: *Corba processes resolves the location of other processes on a network through a Naming Service, before communicating with a new process.*

robot toolkits while mirroring complex CORBA based defence projects such as OCP used on the USAF/USN JUCAV project. This report overviews basic MIRO concepts, describes the Piccolo and Rotomotion autopilots integration efforts. Finally these two methods are reviewed for their respective advantages, disadvantages, and applicability.

1.4 MIRO Design Patterns

The basic software design must bridge the polling functionality of the Piccolo Communi-cation Driver with the subscription model of typical event-based MIRO notifications. MIRO supports three basic design patterns, or design templates of event servicing detailed in the next section.

Though very powerful and robust, CORBA is complex and difficult to exploit, the MIRO framework hides some of this complexity within a small, manageable toolset [9]. A succinct description of MIRO's features and use can be found in [3] As a very brief summary, MIRO exploits five concepts: The Interface Description Language (IDL), The Naming Service, Servers, Clients, and Data Exchange Patterns.

The Interface Description Language describes the properties of data to be passed between processes and the remote methods that operate on them through a C++ or Java-like intermediate language. When compiled, IDL descriptions perform the marshalling and de-marshalling necessary to construct and transmit data types over socket connections. When repeatedly broadcast as a stream, these transmitted data structures are known as events.

The Naming Service is a process that maintains a process database, in effect a 'yellow pages' of locations, of registered CORBA clients and servers. Registration allows other processes to resolve publishers or subscribers of data types and, to a point, removes dependence on particular machine locations.

Servers are usually processes that receive requests and 'serve' information back. MIRO

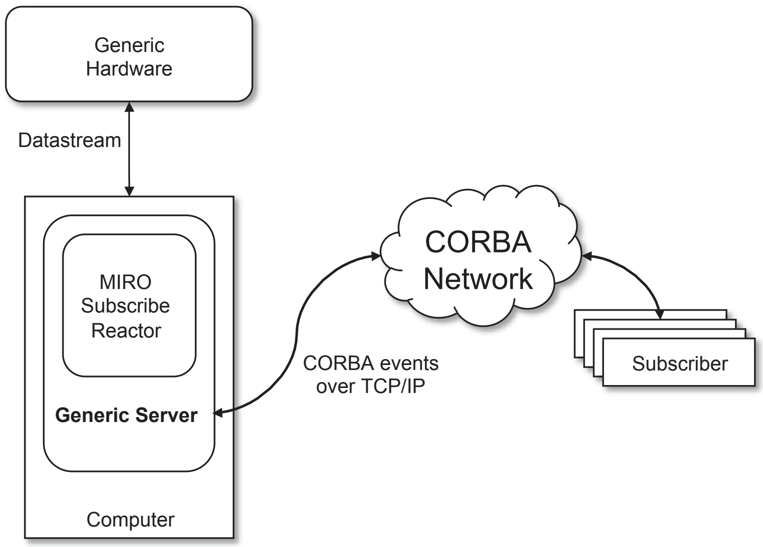


Figure 2: *A typical subscribe component. The reactor handles hardware interrupts, passing the data for parsing and, ultimately, publication over the network.*

Servers publish or broadcast events over an event channel to one or more subscribers. Further the MIRO Server can respond to traditional 'polling' requests for data.

Clients and MIRO Clients are usually processes that generate requests to servers. Historically, this form of unique bilateral communications has been the most common form of interpro-cess communication because of its simplicity. However, the overhead of repeated requests through polling has driven the adoption of ’publish-subscribe’ client-server communications for .

Event Channels and Polling are two forms of interprocess communication. Event Chan-nels provide a vehicle for processes to ’subscribe’ anonymously and asynchronously to data published by the server. Polling is the traditional client-server ask-receive interaction be-tween two processes. Patterns are templates that describe common functional architectures. Miro’s Data Exchange patterns fall into three broad categories:

1. Subscribe-Publish Server allows a MIRO server to receive events, process the data, and publish new events.
2. Subscribe-Reactor Server handles external hardware events, processes and publishes the data as events.
3. Client Patterns capture the typical client server relationship and establish a framework for a process to respond to multiple client requests.

Figure 2 depicts how a ’typical’ subscribe component relates to both the MIRO network and external processes. The subscribe-publish and subscribe reactors sharer identical publish logic, differing only in how the source data is received. Most MIRO processes both receive and transmit MIRO events across event channels and, therefore, fall into the Subscribe-publish pattern. However, sensor input is virtually always taken into the MIRO network through hardware subscribe-reactor drivers.

Subscribe-reactors are unique to the incoming data source. Serial, TCP, and UDP subscribe reactors are designed to manage their respective input transport mechanism. Beneath the transport layer of abstraction each subscribe-reactor must uniquely interact and parse with its partner hardware device, meaning that the programmer must be intimately familiar with the device interface structures and data flow control. Generally, these reactors are configured to place the hardware in a specific set of modes that start, stop, or maintain the data transactions with the device. The MIRO IDL may establish additional controls to switch hardware modes or states — relatively rare for sensors, but common for actuator driven machines.

2 Cloudcap Piccolo II Autopilot

Cloudcap Technologies of Hood River, Oregon, manufactures a line of autopilots based on an MPC555 chipset are depicted in Figure 3. As of 2008, the Piccolo II serves as the core of Cloudcap's product line that includes the Piccolo Plus and Piccolo LT variants. The Piccolo line is one of acommon commercial fixed wing autopilot platforms and has been extended to support helicopter platforms and pan/tilt balls. Since the entire product line is founded on the same Motorola MPC555 chipset, all autopilots possess a common code foundation and communications interfaces. All have GPS (and DGPS), microelectromechanical (MEM) accelerometers and magnetometers, in addition to analog and digital I/O. Each autopilot has a unique radio address, meaning multiple Piccolos can reside and be controlled on the same frequency by a single ground station.

2.1 Piccolo Overview

The Piccolo system is composed of a flying autopilot, a groundstation, an operator interface, and a simulator. Both groundstation and autopilot are composed of an autopilot chipset communicating with one another through a wireless radio link. Each groundstation can address up to 256 aircraft through an internal round-robin wireless networking protocol and each flying Piccolo has a unique address. Cloudcap provides a rudimentary operator interface as a basic vehicle control station and a simple simulator to permit mission practice without autopilot or groundstation hardware.

2.1.1 The Autopilot

The autopilot uses both operating *modes* and configuration *settings* to prescribe different stages of vehicle operation. In effect, the autopilot is a state machine with performance governed by configuration parameters, but driven by the vehicle's sensed state and command modes. Piccolo's 2nd generation controller has the following modes: Prelaunch, Transition, Liftoff, Climbout, Flying, Landing, Final Approach, Short Final, Touchdown, and Rollout. Each of these states maintains control using specific parameter settings and control strategies, many governed by configuration sets and exit conditions into the next state. Flying mode represents the most common running state or 'nominal' mode of the autopilot and all other state are directed towards (e.g. Prelaunch and Launch) or away from (e.g. landing) this mode. In this state, the autopilot gives instructions to the vehicle to follow flightplans composed of a list of waypoints, each representing a desired latitude, longitude and altitude. The autopilot can store 100 waypoints and an arbitrary number of flightplans to a maximum of 50. The waypoint list is represented as a linear array of waypoints each labelled with an array index. Flightplans are, in effect, linked lists of waypoints with each waypoint array entry containing a reference to the next waypoint index. This technique facilitates repeating or circular flightplans.

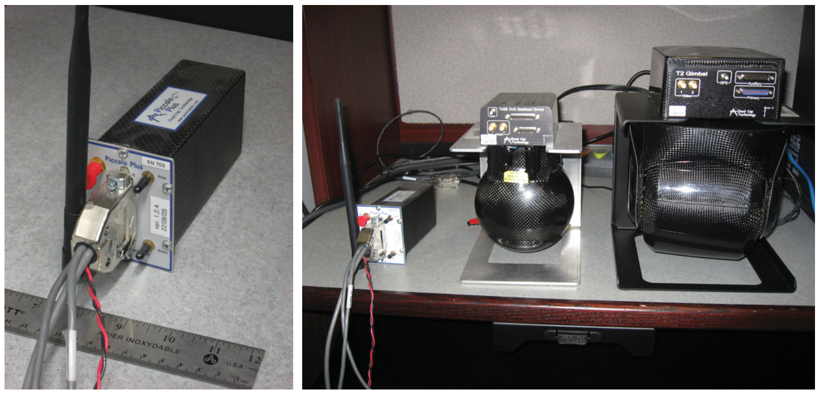


Figure 3: *The older Piccolo Plus (left), used in the ACR Silverfox, conforms to the same API and form factor as the Piccolo II Family, though with less capability. DRDC's current family of Piccolo equipment (right) includes the Plus, the TASE(EO) and TASE Duo(EO/IR) stabilized Piccolo II-based camera balls.*

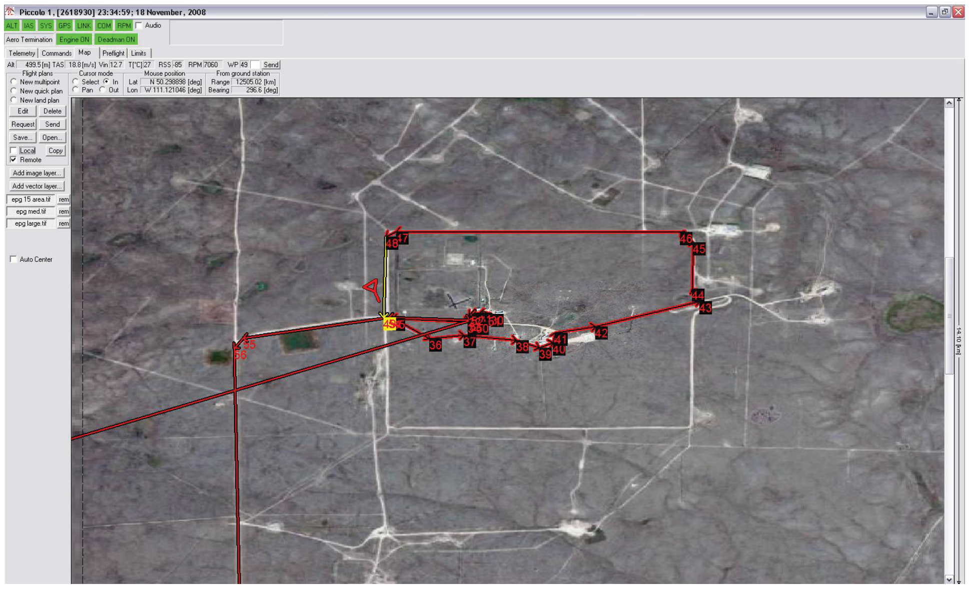
2.1.2 Ground Control Station

The Piccolo architecture exploits a Ground Control Station (GCS) to configure, command, and control all of the flight operations on the airborne Piccolo II. Interestingly, the GCS is itself a Piccolo II configured specifically as a ground control element. A Piccolo appears as a node on an airborne network. The GCS uses round-robin sessions to communicate to each vehicle in the network, a strategy common to other multi-vehicle systems such as DRDC's own ANCAEUS and Micropilot's Horizon autopilot of the same class. The GCS communicates with a simulated Piccolo through a server connection.

2.1.3 Piccolo Operator Interface

To control the GCS, a basic Operator interface known as the Pilot Command Console (PCC), depicted in Figure 4, provides the means to set aircraft parameters, calibrate surfaces, and upload/download waypoint trajectories. Before version 2.1 Cloudcap considered the Piccolo Operator Interface a crude starting point for more serious interface efforts (such as CDL Systems GCS)[1]. Cloudcap provides this software at no cost and a free, though not open, software development kit for communicating to the GCS, vehicle and payload. Though the GCS communication specification is freely available, the extensive instruction set and subtlety of the interface make this SDK an appealing alternative to pure driver development. The GCS communication library facilitates networked connection to the GCS again over a TCP port.

[1]However, with more recent versions, this interface is now marketted as a genuine ground control station and supported by Solutions Engineering Inc.


Piccolo 1, [2618930] 23:34:59; 18 November, 2008

Alt 499.5 [m] TAS 18.8 [m/s] Vin 12.7 T[°C] 27 RSS -85 RPM 7060 WP 49

Mouse position: Lat N 50.298898 [deg] Lon W 111.121046 [deg]

From ground station: Range 12505.02 [km] Bearing 296.6 [deg]

Figure 4: *The Map view using Cloudcap's Operator Interface V1.3.2. Additional Command and Telemetry views permit direct observation and control of vehicle state and track. Red arrow symbols indicates aircraft position and heading, red line segments indicate stored tracks, and yellow indicates the current desired track.*

2.1.4 Aircraft Simulator

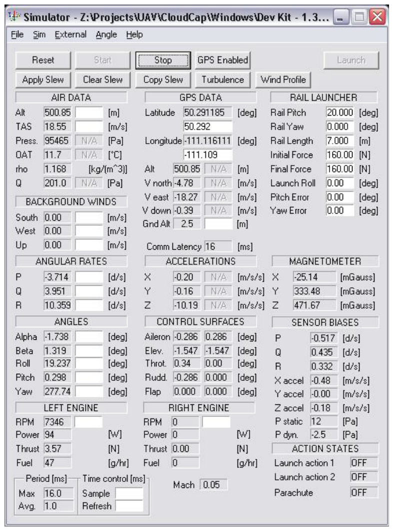
To rehearse vehicle missions, Cloudcap provides a configurable dynamic simulator (as shown in Figure 5) that simulates the motion of a user described vehicle and generates both sensor inputs to the Piccolo II and full vehicle state to data subscribers. User's can fully configure a selection of aerodynamic, actuator, and sensor properties to describe an aircraft of their design. The simulator produces sensor outputs (e.g. pitot-static pressures), actuator commands, IMU sensing, and GPS consistent with this model, responding to control surface commands appropriately. Further, the simulator broadcasts the kinematic vehicle state over a UDP connection to FlightGear clients, greatly simplifying vehicle rendering.

2.1.5 Piccolo Simulator

To permit full software simulation of a UAV system, Cloudcap has produced a simulated version of the autopilot, `PiccoloPC`. This simulator is identical in all respects to the hardware version of the Piccolo, but receives and broadcasts state information over TCP ports.

2.1.6 GroundStation Simulator

The simulation system permits developers to simulate all components of the system, including the groundstation, `GroundStationPC`. This command-line service mimics the behaviour


Simulator - Z:\Projects\UAV\CloudCap\Windows\Dev Kit - 1.3...
File Sim External Angle Help
Reset
Start
Stop
GPS Enabled
Launch
Apply Slew
Clear Slew
Copy Slew
Turbulence
Wind Profile

```
AIR DATA                          GPS DATA                              RAIL LAUNCHER
Alt     500.85          [m]       Latitude  50.291185         [deg]     Rail Pitch     20.000  [deg]
TAS     18.55           [m/s]               50.292                      Rail Yaw       0.000   [deg]
Press.  95465   N/A     [Pa]      Longitude -111.116111       [deg]     Rail Length    7.000   [m]
OAT     11.7    N/A     [°C]                -111.109                    Initial Force  160.00  [N]
rho     1.168     [kg/(m^3)]      Alt       500.85   N/A      [m]       Final Force    160.00  [N]
Q       201.0   N/A     [Pa]      V north   -4.78    N/A      [m/s]     Launch Roll    0.00    [deg]
                                  V east    -18.27   N/A      [m/s]     Pitch Error    0.00    [deg]
BACKGROUND WINDS                  V down    -0.39    N/A      [m/s]     Yaw Error      0.00    [deg]
South   0.00            [m/s]     Gnd Alt   2.5               [m]
West    0.00            [m/s]
Up      0.00            [m/s]     Comm Latency 16             [ms]

ANGULAR RATES                     ACCELERATIONS                         MAGNETOMETER
P       -3.714          [d/s]     X         -0.20    N/A      [m/s/s]   X   -25.14   [mGauss]
Q       3.951           [d/s]     Y         -0.16    N/A      [m/s/s]   Y   333.48   [mGauss]
R       10.359          [d/s]     Z         -10.19   N/A      [m/s/s]   Z   471.67   [mGauss]

ANGLES                            CONTROL SURFACES                      SENSOR BIASES
Alpha   -1.738          [deg]     Aileron   -0.286   0.286    [deg]     P          -0.517  [d/s]
Beta    1.319           [deg]     Elev.     -1.547   -1.547   [deg]     Q          0.435   [d/s]
Roll    19.237          [deg]     Throt.    0.34     0.00     [deg]     R          0.332   [d/s]
Pitch   0.298           [deg]     Rudd.     -0.286   0.000    [deg]     X accel    -0.48   [m/s/s]
Yaw     277.74          [deg]     Flap      0.000    0.000    [deg]     Y accel    -0.00   [m/s/s]
                                                                        Z accel    -0.18   [m/s/s]
LEFT ENGINE                       RIGHT ENGINE                          P static   12      [Pa]
RPM     7346                      RPM       0                           P dyn.     -2.5    [Pa]
Power   94              [W]       Power     0                 [W]       ACTION STATES
Thrust  3.57            [N]       Thrust    0.00              [N]       Launch action 1    OFF
Fuel    47              [g/hr]    Fuel      0                 [g/hr]    Launch action 2    OFF
                                                                        Parachute          OFF
Period [ms]    Time control [ms]  Mach  0.05
Max   16.0     Sample
Avg.  1.0      Refresh
```

Figure 5: *The Simulator GUI depicting the Aircraft State (in this case an R/C Cub sen-sor/actuator model).*

of the groundstation hardware and software and can communicate with a genuine or simulated Piccolo through serial or network ports. Similarly, communication with the Operator interface continues transparently through a dedicated network port.

2.2 Communications Interface

The Piccolo communicates vehicle state and accepts commands through the GCS, itself a Piccolo. The Piccolo GCS, in turn, communicates to a host computer through a serial port (RS-232) connection. PC based software can communicate with the flying Piccolo either through directly decoding the Piccolo GCS packet stream or using Cloudcap's native serial driver. The PCC uses Cloudcap's native drivers to communicate with the GCS, but can also package unaltered data streams direct to a socket connection. Thus developers seeking to further increase mission autonomy through smarter ground control have three alternatives to marshall/demarshall Piccolo GCS data streams:

- develop custom Piccolo serial drivers
- use Cloudcap's Piccolo serial drivers
- use Cloudcap's Piccolo network drivers combined with the PCC.

The Piccolo uses a hierarchy of data structures to encapsulate the function of the autopilot. Perhaps the most important or central data structure is the `CloudCapUserData` structure.

2.3 Software Design

To add mission capabilities with the least risk, DRDC Suffield implemented MIRO compo-nents using the third option: network drivers with PCC. The development of custom serial drivers is common practice for robotics toolkits, for example DRDC's MIRO gps and imu drivers – an approach that reduces transaction processing overhead, but greatly increases the developer's responsibility to maintain code. Using Cloudcap's linux serial drivers would reduce this burden but would render the PCC unusable on the same GCS. The solution is to use the network driver embedded in the Cloudcap SDK. This driver exposes the full vehicle mission control interface to a network client without displacing the PCC – an unavoidable outcome of a direct serial connection with the GCS. In short, experimental mission control software can be tested with proven ground control software acting as safety oversight. Given the number and variety of air, land, and marine autopilots and the profusion of interfaces, the ability to incorporate these devices into the more sophisticated MIRO framework will greatly expand the scope of MIRO control.To understand the final design approach, requires an examination of the Cloudcap communication driver, the options for mission autonomy, and the limits of the aircraft.

2.3.1 Piccolo Generator Component: *PiccoloServer*

The typical Miro driver, like the one depicted in Figure 6 could replace not only the Pic-colo Communication Driver, but the PCC completely. This approach requires parsing the Piccolo serial data stream and mimicking the PCC's management of Autopilots. Though the structures are large and demarshalling the packets complex, the Piccolo's serial data structure is publicly available through published APIs and, therefore, demarshalling is dif-ficult, but achievable. However removing or replacing the PCC places the vehicle at risk. The PCC provides both extensive visual warnings to operators and graphically represents mission progress on a moving map display. Removal of this display without replacement is not reasonable until a MIRO based ground station becomes available. Cloudcap antici-pated the need for continuous PCC control during development and subsequently provided a network interface to the PCC to which clients could post requests. In effect the PCC broadcasts the autopilots datastream over a UDP socket connection. Clients on this socket make requests to the PCC's server and receive packets that closely resemble the native serial packets arriving over the serial port from the GCS. This relationship is depicted in Figure 7.

This approach retains the PCC in the command loop while providing developers with an unmodified connection to the autopilot. However, the marshalling and demarshalling of data remains a problem with developers having to decode the incoming stream, manage errors, and maintain the Piccolo network model. So while this model retains pilot control and MIRO provides the `SocketConnection` Reactor, development remains substantial. Again, Cloudcap anticipated this impediment and provides a Software Development Kit (SDK) that marshalls/demarshalls and performs basic autopilot network housekeeping. To merge MIRO with the SDK means creating and maintaining a Cloudcap autopilot client within some form of Timed Publish pattern.

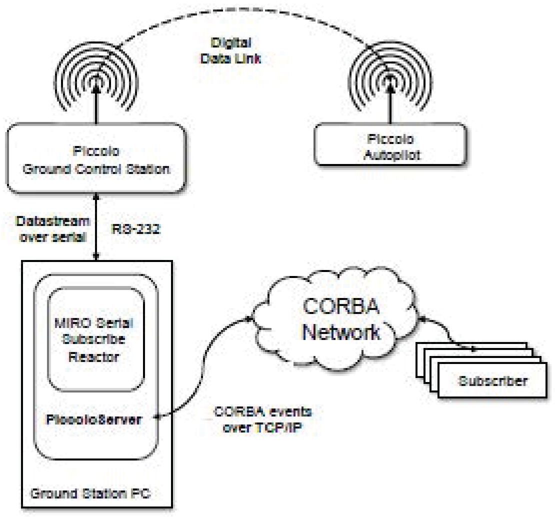


Figure 6: *Typical* MIRO *approach replaces the PCC with a* MIRO *Reactor as the primary vehicle communication module.*

2.3.2 Timed Publish Poll

The resulting *PiccoloServer* depicted in Figure 8 is a variant of the Subscribe-Reactor [2]. The Subscribe-Reactor of Figure 3 responds to hardware events, such as activity on the serial port, to then generate a CORBA event. However, in some cases hardware devices must be polled to generate events. A good example is the Microstrain 3dmg that will not produce quaternion, homogeneous matrix, and RPY representations in a continuous stream. The solution is to prompt the 3dmg with round-robin polling for these values, finally passing the cumulative result as a single IMU event. The resulting design pattern, a Publish-Poll is thus a simplified variant of the basic Publish-Reactor.

The Piccolo generator depicted in Figure9 adopts the basic structure of the Publish-Poll, but exchanges round-robin triggering for timers to both sample the Piccolo Driver and issue timed updates on the Piccolo state. The result is a Publish-Timed-Poll containing two timers, one devoted to poll sampling of the Piccolo data stream and the other devoted to updating the Publish stream as depicted in Figure 9.

2.3.3 Cloudcap IDLs

Piccolo Events are published (and polled) through the Cloudcap IDL. The IDL is composed of a data portion, describing data structure candidates used in either polled or published data exchanges, and an interface part, describing methods available to client processes. The data structures substantially mirrors the data structure of the original serial data stream, though actual data types are not necessarily identical due to differences between CORBA and Piccolo integer and floating point definitions.

[2]really a misnomer for Publish-Reactor, since the component reacts to hardware and *publishes* events

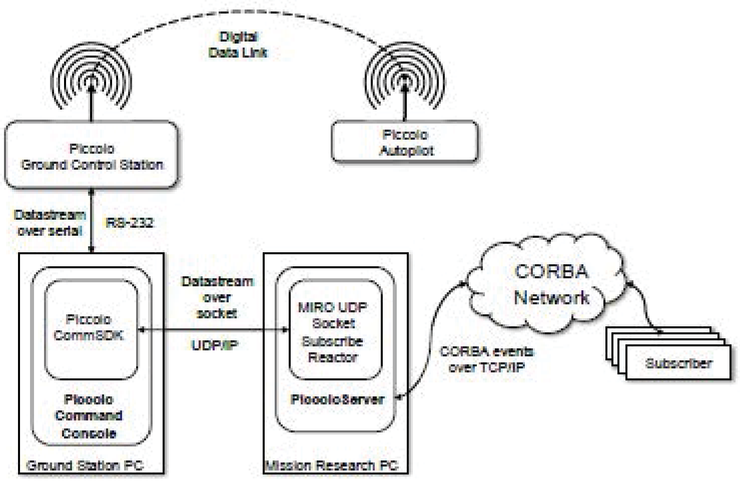


Figure 7: *One solution augmenting the PCC with custom UDP* MIRO *components that maintains the GCS in the loop.*

The interface part permits clients of the IDL event stream to retrieve and control the publication frequency. Though the IDL describes data candidates for publication, only those data types actually published in the component are available for subscription.

PiccoloServer currently publishes three data structures: `Pose`, `CloudCapUserDataIDL`, and `CloudCapAutopilotCmd`. The Pose IDL describes a basic position and orientation in space through an Homogeneous Transform with respect to UTM coordinates. `CloudCapUserDataIDL` is a large hierarchical data structure essentially similar to Piccolo's Communication SDK `UserData_t` structure. `CloudCapAutopilotCmd` encapsulates commands sent by the operator to and echoed by the piccolo. Reactor based components that must decode an external hardware stream will often simply rebroadcast the native interface as an IDL. This strategy serves two purposes: rebroadcasting the device interface greatly speeds component design and development and produces data consistent with device documentation albeit through a CORBA medium. However, such expediency reveals hardware details into the MIRO information flow – violating information hiding principles. At a more abstract level, the information flowing from the device should be generalized away from a specific product towards a generic 'autopilot' IDL, for example.

2.3.4 An Example Subscriber: qtEventPiccolo

To test both the `CloudCapUserDataIDL` and `CloudCapAutopilotCmd` IDLs, a Qt-based graphical test module was written, `qtEventPiccolo` shown in Figure 10. As mentioned earlier, the PiccoloServer generates events based on timers. With these two IDL streams to service, PiccoloServer uses two timers to govern the publication of events. Therefore, both IDLs provide a poll interface mechanism to permit clients to set the timer update rate. `qtEventPiccolo` demonstrates this capability through a simple timer setting GUI element that permits users to observe the domain, type, and timestamp of arriving events

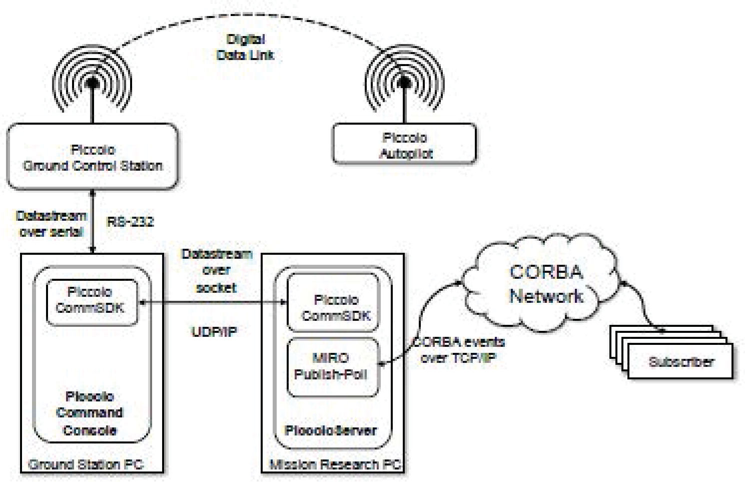


Figure 8: *A solution augmenting the PCC with an SDK-based* MIRO *component that retains the GCS in case of failure.*

and to change the update rates for each event type. A sample of Autopilot state data contained in the `CloudCapUserDataIDL` is updated at the assigned rate, specifically the the position and heading of the Piccolo Autopilot. The client automatically downloads the entire Waypoint memory of the autopilot, and uses the `CloudCapAutopilotCmd` to retrieve the current tracking waypoint. Using this basic UI client as a foundation, additional capabilities can be added including waypoint editing, moving map displays, and low level vehicle teleoperation.

`qtPoseEvent`, another graphical service, permits inspection of the published Pose IDL stream.

2.4 Visualization

MIRO and Cloudcap make a number of visualization options possible. Three notable avenues are direct Simulator to FlightGear rendering, indirect Groundstation to Gazebo rendering, and still more removed Groundstation to Google Earth.

2.4.1 Simulator to FlightGear

The Piccolo command console can direct aircraft state information via socket connections to the FlightGear [1] open source flight simulator. Since FlightGear's API permits external sources to feed and update the location of an aircraft, Cloudcap provides the simulator with a mechanism to represent the a Piccolo-controlled aircraft in Flightgear. This approach works very well. However, since the simulator is not used during actual flight this approach has little practical value. Though a similar method could be developed using MIRO to drive the rendering, FlightGear's lack of solid mechanics simulation makes this only a short term solution.

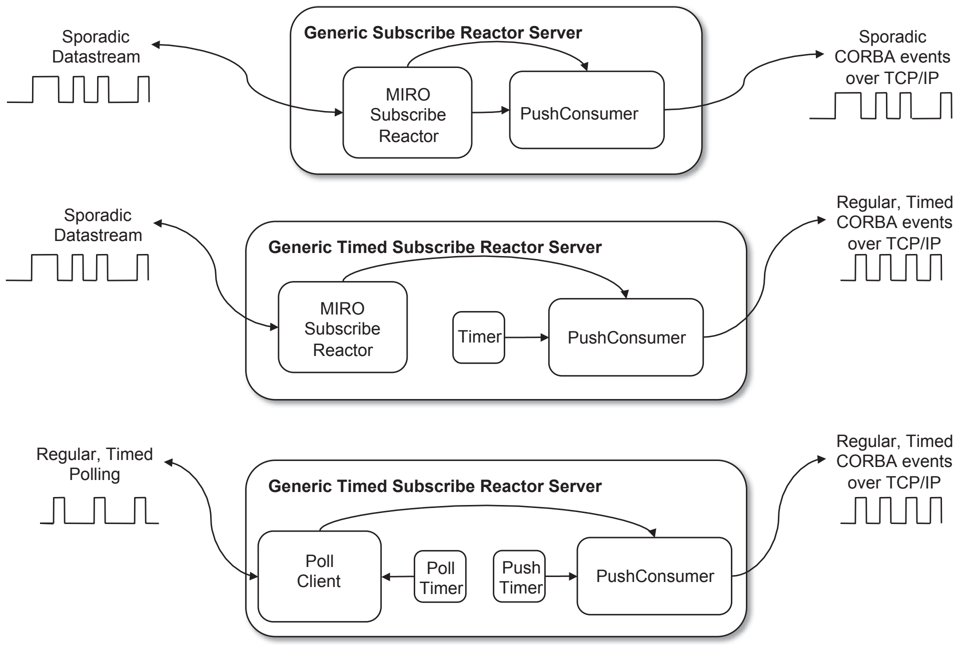

Sporadic Datastream
Generic Subscribe Reactor Server
MIRO Subscribe Reactor
PushConsumer
Sporadic CORBA events over TCP/IP
Sporadic Datastream
Generic Timed Subscribe Reactor Server
MIRO Subscribe Reactor
Timer
PushConsumer
Regular, Timed CORBA events over TCP/IP
Regular, Timed Polling
Generic Timed Subscribe Reactor Server
Poll Client
Poll Timer
Push Timer
PushConsumer
Regular, Timed CORBA events over TCP/IP

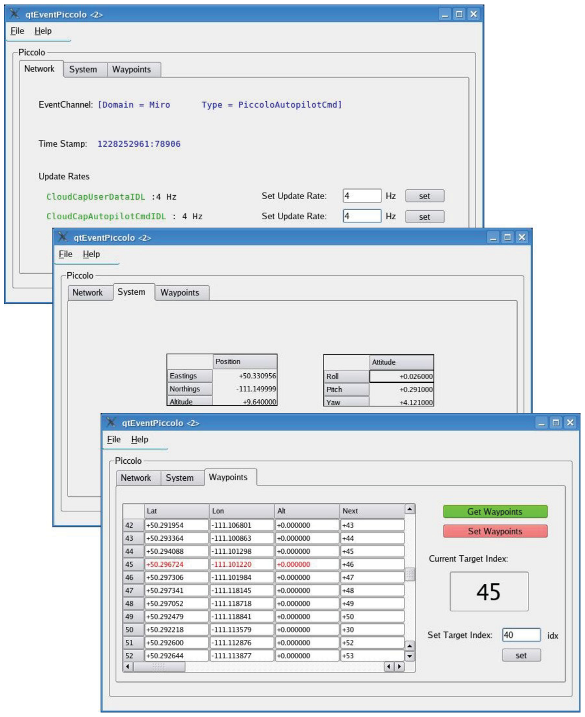
Figure 9: *The relationship between pure event-driven Subscribe-Reactor, Timed Reactor, and Timed Publish-Poll components.*

2.4.2 Simulator to Gazebo

Gazebo is a popular 3D dynamic modeling and rendering environment used in conjunction with the Player [5], a 'device server' and simulation toolkit. Gazebo is founded on the *Open Dynamics Engine* (ODE), an open source rigid body mechanical simulator. Like all rigid body simulators, ODE integrates the kinematic state of a virtual mechanical 'world', as a set of constrained rigid bodies, to ultimately produce the body positions. By applying virtual forces to these bodies, mechanisms can move and interact with a virtual environment. Gazebo wraps ODE with a simplified and dedicated device list that both inserts rigid body components into the world and samples the world to produce synthetic sensor data. Clients can call upon Gazebo services through a traditional client server model.

Though Player/Stage/Gazebo provides a no-cost entry to robot simulation and/or rendering, Gazebo's ODE does not support Computational Fluid Dynamics (CFD). Therefore, either an external aerodynamic simulator must be used to simulate an autopilot controlled UAV or a live/logged UAV state must be provided to the renderer. Though, Cloudcap and Rotomotion do provide a simple simulator for training, both lacks the exhaustive simulation capabilities of systems such as FlightGear [1]. Thus building a Simulator/Gazebo adaptor could provide the means of either simulating or merely viewing vehicle motion in a rendered environment.

A primitive polling client to `Gazebo`, `PlayerBase`, is provided with the base MIRO release. Using the Piccolo IDL in conjunction with the `PlayerBase` MIRO API, a UAV location can be



EventChannel: [Domain = Miro Type = PiccoloAutopilotCmd]

Time Stamp: 1228252961:78906

Update Rates

CloudCapUserDataIDL :4 Hz — Set Update Rate: 4 Hz set

CloudCapAutopilotCmdIDL : 4 Hz — Set Update Rate: 4 Hz set

	Position
Eastings	+50.330956
Northings	-111.149999
Altitude	+9.640000

	Attitude
Roll	+0.026000
Pitch	+0.291000
Yaw	+4.121000

	Lat	Lon	Alt	Next
42	+50.291954	-111.106801	+0.000000	+43
43	+50.293364	-111.100863	+0.000000	+44
44	+50.294088	-111.101298	+0.000000	+45
45	+50.296724	-111.101220	+0.000000	+46
46	+50.297306	-111.101984	+0.000000	+47
47	+50.297341	-111.118145	+0.000000	+48
48	+50.297052	-111.118718	+0.000000	+49
49	+50.292479	-111.118841	+0.000000	+50
50	+50.292218	-111.113579	+0.000000	+30
51	+50.292600	-111.112876	+0.000000	+52
52	+50.292644	-111.113877	+0.000000	+53

Get Waypoints
Set Waypoints

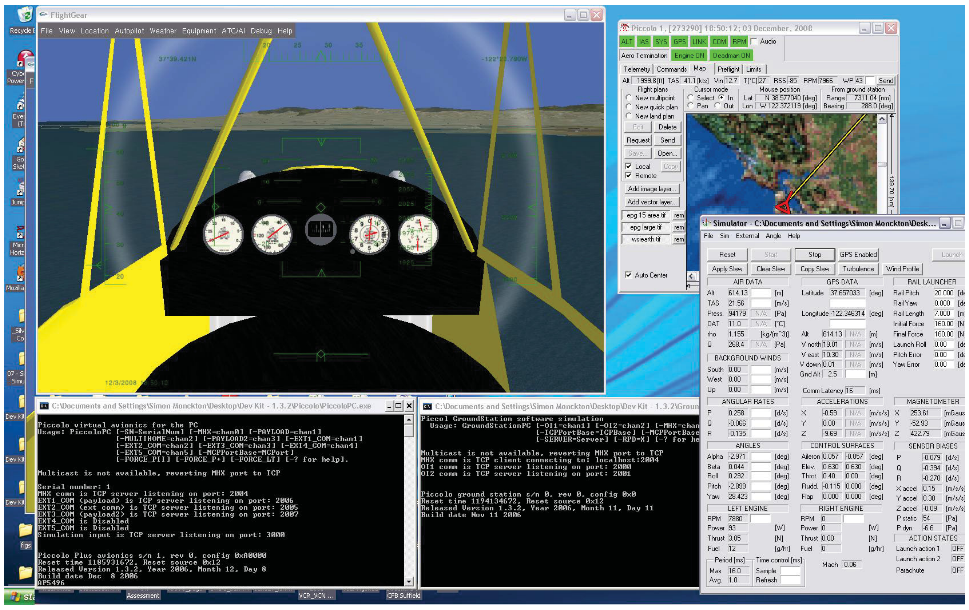
Current Target Index: 45

Set Target Index: 40 idx set

Figure 10: *The Network tab (top) contains poll interface controls for both* `CloudCapUserDataIDL` *and* `CloudCapAutopilotCmd` *IDL update rate settings. The System tab (middle) reports on the vehicle position, while the Waypoint tab (bottom) provides the waypoint list and current waypoint.*

sent to the Player/Stage/Gazebo robot simulator for visualization. Though a useful starting point, PlayerBase proved to make vehicle rendering an awkward process and was rewritten by DRDC Suffield into GazeboServer. This direct connection to Gazebo replaces the player client with a MIRO reactor task. This service may provide improve engine performance in the rendering role. Though more successful than PlayerBase, GazeboServer cannot correct the basic problem in Gazebo that externally driven bodies have no mass properties. Thus vehicles rendered through this mechanism tend to 'snap' from position to position based on external simulator updates and do not move smoothly as expected.

Despite this deficiency, Gazebo offers a simple open and widely supported rendering system and could provide the core foundation elements for a federated simulation system. A federated system employs a dynamic modeling engine for each physical environment (e.g. air, water, and ground). With this future in mind, `ModelServer` and its supporting assembly and body modeling file formats, were modified to provide limited Support for Gazebo. ModelServer relies on a tree-like structure to reflect the typical hierarchical relationship between discrete mechanical components on a robot. For example the relationship between the cameras, GPS antenna, and autopilot on a UAV can be related to a fixed coordinate system on the fuselage. Thus a typical ModelServer Assembly file for a Silverfox contains airframe and component geometry data. `ModelServer` can export this data into a simpli-



```
C:\Documents and Settings\Simon Monckton\Desktop\Dev Kit - 1.3.2\Piccolo\PiccoloPC.exe
Piccolo virtual avionics for the PC
Usage: PiccoloPC [-SN=SerialNum] [-MHX=chan0] [-PAYLOAD=chan1]
                 [-MULTIHOME=chan2] [-PAYLOAD2=chan3] [-EXT1_COM=chan1]
                 [-EXT2_COM=chan2] [-EXT3_COM=chan3] [-EXT4_COM=chan4]
                 [-EXT5_COM=chan5] [-MCPPortBase=MCPort]
                 [-FORCE_PII] [-FORCE_P+] [-FORCE_LT] [-? for help].

Multicast is not available, reverting MHX port to TCP

Serial number: 1
MHX comm is TCP server listening on port: 2004
EXT1_COM (payload) is TCP server listening on port: 2006
EXT2_COM (ext comm) is TCP server listening on port: 2005
EXT3_COM (payload2) is TCP server listening on port: 2007
EXT4_COM is Disabled
EXT5_COM is Disabled
Simulation input is TCP server listening on port: 3000


Piccolo Plus avionics s/n 1, rev 0, config 0xA0000
Reset time 1185931672, Reset source 0x12
Released Version 1.3.2, Year 2006, Month 12, Day 8
Build date Dec  8 2006
AP5496
```

```
C:\Documents and Settings\Simon Monckton\Desktop\Dev Kit - 1.3.2\Groun
Piccol GroundStation software simulation
  Usage: GroundStationPC [-OI1=chan1] [-OI2=chan2] [-MHX=chan
                         [-TCPPortBase=TCPBase] [-MCPPortBase
                         [-SERVER=Server] [-RPD=X] [-? for he

Multicast is not available, reverting MHX port to TCP
MHX comm is TCP client connecting to: localhost:2004
OI1 comm is TCP server listening on port: 2000
OI2 comm is TCP server listening on port: 2001


Piccolo ground station s/n 0, rev 0, config 0x0
Reset time 1194134672, Reset source 0x12
Released Version 1.3.2, Year 2006, Month 11, Day 11
Build date Nov 11 2006
```

Figure 11: *Flightgear accepts state assignment from external simulation engines. The position and orientation of the vehicle is applied to a vehicle 'skin' through an established Flightgear API.*

fied Gazebo world file structure and provides a rudimentary structure that Gazebo can be render. More elaborate rendered structures are possible through Gazebo's Mesh support that permits a 'skin' to be applied to basic body components.

2.4.3 Ground Station to Google Earth (GE)

Google Earth is a free online imaging and mapping utility provided by Google Inc. This software provides users with the capability to mark points of interest, paths, and regions on satellite imagery. Further, this imagery is laid over digital terrain and may be viewed from arbitrary perspectives to clarify detail in altitude. Able to periodically poll or sample files or network services, GE has been used to present vehicle movement in a number of applications [8].

GE maintains a continuous connection through a *network link* composed of two files. The first describes the network connection and protocol to sample the second file. The second file contains the sampled data. Once loaded into GE, the network link is sampled periodically over an HTTP connection. Simultaneously, another process can write to this file to update the data. The Piccolo Generator provides an option for periodically sampling network link files describing one or more vehicle positions. In Figure 12, the following description file for a fixed wing Silverfox is updated through a MIRO driver:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">

<NetworkLinkControl>
    <Update>
        <targetHref>http://enkidu/KML/NetworkLinkTest.kml</targetHref>
            <Change>
                <Placemark targetId="pm123">
                    <name> SF1 0.00 m:0.00 deg</name>
                    <Point>
                        <coordinates>0.000000,0.000000,0.000000</coordinates>
                    </Point>
                </Placemark>
            </Change>
    </Update>
</NetworkLinkControl>
</kml>
```

`PiccoloServer` periodically writes the Sampled Data file. An example:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
<Document>
    <Style id="FriendlyFWUAV">
        <IconStyle>
            <scale>0.5</scale>
            <Icon>
                <href>http://enkidu/KML/UAV-FixedWing-friendly.bmp</href>
            </Icon>
        </IconStyle>
    </Style>
    <Placemark id="pm123">
        <name>SilverFox01</name>
        <styleUrl>#FriendlyFWUAV</styleUrl>
        <Point>
            <coordinates>-111.112897,50.298282,609.270000</coordinates>
        </Point>
    </Placemark>
</Document>
</kml>
```

This same interface can be used to present any vehicle data desired. Unfortunately, the Network Links facility is limited to data sampling only. Google Earth requires AJAX techniques for richer interaction with other services. GE can sample the Network Link at no faster than 1Hz. Practically, however, high speed sampling increases the probability of a file block condition between the PiccoloServer and GE. Though GE ignores with blocked links, 1 Hz sample rates cannot be gauranteed if the Server is writing at a similar rate.

3 Rotomotion Autopilot

Rotomotion provides an experimental UAV autopilot with roots in an open source project (http://autopilot.sourceforge.org) for rotorwing aircraft. The Rotomotion Autopilot is

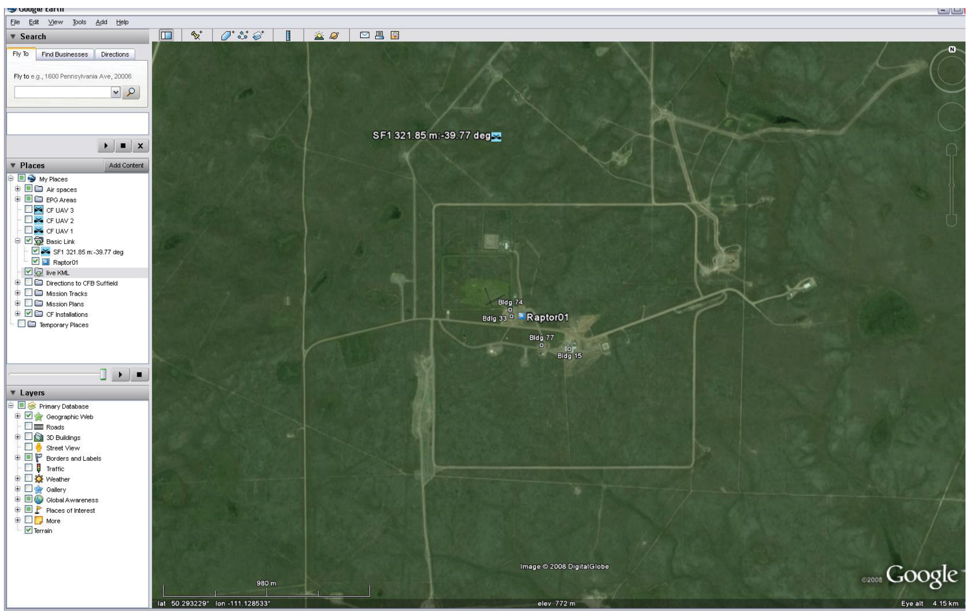


Figure 12: *Using Network Links Google Earth can provide a simple low cost method of observing vehicle tracks over a network. Vehicle Control, however, will require greater access to the Google Earth API.*

based on a Linux embedded computer chipset and has flown at least three scales of helicopter in the 5 to 25kg payload range (the SR20, SR100, and SR200). The SR20 is an all-electric and both SR100 and SR200 are gas-powered helicopter.

3.1 Ground Control Station

Like the Piccolo II, the Rotomotion Ground Control Station (GCS) depicted in Figure 13 may be used to configure, command, and control all of the flight operations on the airborne Autopilot. Since all the vehicles in a Rotomotion network use basic IEEE 802.11b TCP/IP communications, the GCS is merely a client process running on either a Windows or Linux desktop computer. Each Rotomotion autopilot appears as a node on an airborne network. Communication scheduling between air and ground is left to the wireless router and is transparent to the user, an interesting strategy that delegates communications to international standards, but remains limited to short range line of sight.

The ground station comes with a crude graphical user interface (GUI) for interacting with the autopilot on the UAV. The flight computer (or autopilot) communicates with the ground control software (GCS) through a network using the User Datagram Protocol (UDP) protocol from the TCP/IP family. The relatively uncertain wireless communication environment and the resulting reduced overall bandwidth makes UDP a better choice for air-ground communications.

The flight computer acts as the server and the GCS as a client connecting to the server.

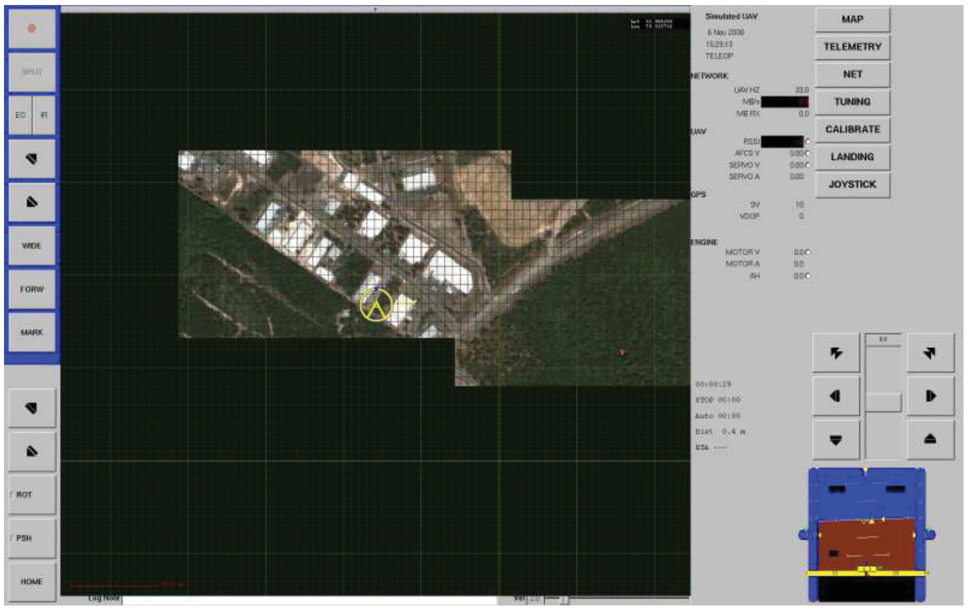


Figure 13: *The Rotomotion GCS software is relatively crude compared to the Cloudcap PCC, though the basic client server approach is arguably more polished.*

The server provides information such as current attitude, current position, currently desired attitude, currently desired position, raw sensor data, video feed and other. Logging data is done on the client side. The GUI is compatible with both Windows and Linux platform. The ground station software allows the user to program the waypoint mode and to see the realtime status of the UAV systems via a moving map display, an artificial horizon and other multifunction display panels (MFD) such as status, servo commands, etc. The radio link between the GCS and the flight controller is done with two wireless routers.

3.2 Aircraft Simulator

To test vehicle performance, Rotomotion provides a simple dynamic simulator that generates sensor inputs to the autopilot and full vehicle state to data subscribers. The simulator produces sensor outputs such as altimetry, control servos, IMU sensing, and GPS consistent with a flying model of the user's design and responds to control surface commands appropriately.

3.3 Autopilot

The autopilot hardware, shown in Figure 15 is an integrated package designed to control and guide an R/C-class helicopter. A 1.25lb avionics unit is mounted to the helicopter and the control parameters tuned for the specific airframe. An additional 7.2V battery is required to run the guidance system. The servo driver, safety over-ride and receiver are powered by a separate conventional 4.8V R/C RX battery.

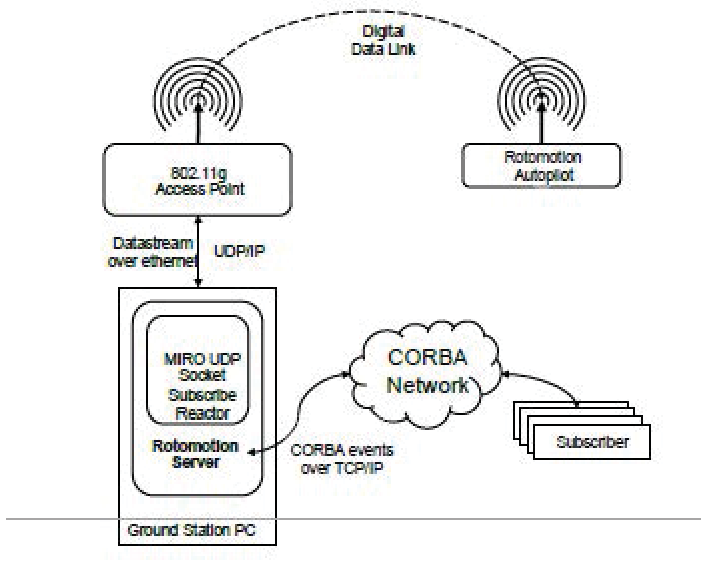


Figure 14: *The Rotomotion driver adopts a more traditional MIRO approach, replacing the GCS with MIRO components as the primary vehicle control station.*

The operator may either take-off and land the helicopter manually or use auto take-off and auto-landing modes. Once the helicopter airborne and placed into a hover, a toggle switch on the transmitter shifts the helicopter into autonomous mode. At this point, the helicopter no longer requires direct manual control. The autonomous flight control system uses an advanced stable-hover control system. Out of hover, the helicopter has a number of path following modes:

- Velocity command mode (VC-Mode): the transmitter controls position through proportional velocity commands. For example, the cyclic control stick becomes the velocity control stick in velocity command mode. The stick commands the helicopter to move in the command direction at a speed proportional to the amount of stick movement on the transmitter.
- Waypoint route mode (WAY-Mode) : the vehicle flies a preprogrammed series of waypoints
- Fast forward flight mode (FFF-Mode) : the helicopter flies in fast forward flight similar to a fixed wing aircraft and, since forward flight is less power consumptive, the mode can force the helicopter to orbit an area, increasing flight efficiency.

3.3.1 Flight Controller

The Rotomotion Automatic Flight Control System (AFCS) consists of five modular components :

1. 3 axis, 6 degrees of freedom inertial measurement unit (IMU)

2. 3 axis magnetometer
3. GPS with optional DGPS receiver
4. Proprietary radio receiver with servo interface and safety pilot override
5. Linux-based flight computer

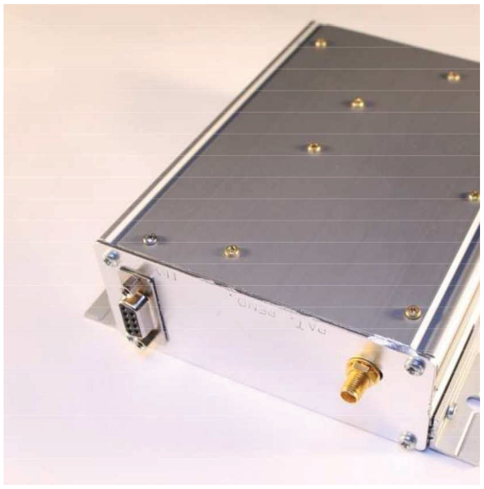


Figure 15: *A view of the Rotomotion autopilot revealing onboard I/O, GPS antennas, and basic construction.*

3.4 MIRO Implementation

Though Rotomotion does not provide driver to decode incoming messages, supplied documentation describes each message structure. Since MIRO natively supports the TCP protocol, the MIRO Subscribe-Reactor was modified to support UDP messages. By electing to create code to parse Rotomotion messages, DRDC assumes responsibility for the tracking any changes to the autopilot's software implementation. Any change in message structure must be mirrored DRDC software.

Depending on an autopilot's design, a MIRO driver might not be able to run alongside the ground control station (GCS). Fortunately, the Rotomotion autopilot and GCS have been designed using client-server principles, meaning the server (autopilot) accepts more than one client connection so we can simultaneously run GCS and MIRO services. The resulting structure is depicted in Figure 14 If this were not the case and the server could not accept more than one connection, a "server-in-the-middle" (as in Figure 16) could be devised which would accept at least two connections (one from a GCS software and one from a MIRO service) and connect to the "real" server and broadcasting incoming data to all clients.

Autopilots that use ethernet protocols for air-to-surface communication must use either TCP or UDP protocols. This section will explore the advantages and disadvantages of these respective protocols. CORBA exploits TCP as the low level, *transport layer* for interprocess communications.

3.4.1 Transmission Control Protocol (TCP)

The transmission control protocol (TCP) is a *connection-oriented* protocol intended for reliable transmission between two processes. This means that one end point needs to open a *socket* on the machine/computer/device and initiates a connection with the other end point (3-way handshake). A socket or network socket is an interface between an application and the TCP/IP *protocol stack*. The term "connection-oriented" implies that an initialization step establishes a continuous relationship between sender and receiver. Usually, end points are called respectively client and server, where the client process submits requests to and receives answers from a server process.

TCP is *reliable* because it guarantees no packet loss, error-free data transfer, discarding duplicate packets and packet ordering for example. To avoid packet loss, an acknowledgement system is used where a receiving process transmits an ACK message to the transmitting process for every message received meaning the receiver is ready for another data transfer. TCP also provides flow and congestion control. Flow control is implemented to avoid having the sender send data too fast for the TCP receiver to reliably receive and process it. Congestion control manages traffic on a network to avoid oversubscription of any of intermediate network nodes by taking resource reducing steps, such as reducing the packet transmission rate. Each TCP header size is at least 20 bytes long without the options and data.

3.4.2 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is a *connectionless* protocol that provides unreliable transmission and with very few error recovery services. It's primarily used for *broadcasting* messages over a network. UDP does not guarantee reliability or ordering in the way that TCP does. Datagrams may arrive out of order, appear duplicated, or go missing without notice. Avoiding the overhead of checking whether every packet actually arrived makes UDP faster and more efficient, for applications that do not need guaranteed delivery. The size of a UDP packet is at least 8 bytes long without the data.

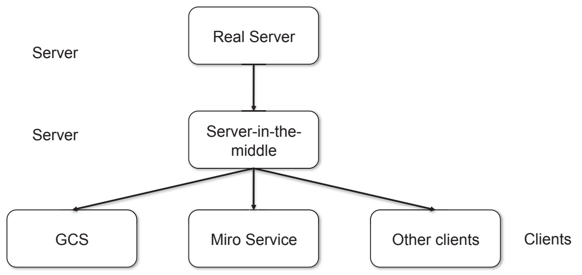


Figure 16: *Server-in-the-middle structure*

3.4.3 Difference between TCP and UDP

TCP is a connection-oriented protocol, which means that upon communication it requires 3-way handshake to set up end-to-end connection.

- Reliable - TCP manages message acknowledgment, retransmission and timeout. Many attempts to reliably deliver the message are made. If it gets lost along the way, the server will re-request the lost part. In TCP, there's either no missing data, or, in case of multiple timeouts, the connection is dropped.
- Ordered - if two messages are sent along a connection, one after the other, the first message will reach the receiving application first. When data packets arrive in the wrong order, the TCP layer holds the later data until the earlier data can be rearranged and delivered to the application.
- Heavyweight - TCP requires three packets just to set up a socket, before any actual data can be sent. It handles connections, reliability and congestion control.
- Streaming - Data is read as a stream, with nothing distinguishing where one packet ends and another begins. Packets may be split or merged into bigger or smaller data streams arbitrarily.

UDP is a simpler message-based connectionless protocol. In connectionless protocols, there is no effort made to setup a dedicated end-to-end connection. Communication is achieved by transmitting information in one direction, from source to destination without checking to see if the destination is still there, or if it is prepared to receive the information.

- Unreliable - When a message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission and timeout.
- Not ordered - If two messages are sent to the same recipient, the order in which they arrive cannot be predicted.
- Lightweight - There is no ordering of messages, no tracking connections, etc.
- Datagrams - Packets are sent individually and are guaranteed to be whole if they arrive. Packets have definite bounds and no split or merge into data streams may exist.

3.4.4 Design contraints

Usually, the weakest link between a ground control station (GCS) and an aircraft is the wireless link. The available bandwidth and the subsequent volume of data transfer depends significantly on the link transmitters, vehilce operating altitude, antenna propertie — all often bounded by the vehicle payload. By choosing a UDP protocol over a TCP, one can eliminate or reduce the overhead due to error recovery, handshakes, packet ordering and packet retransmission. However, by opting for UDP, the autopilot needs to implement

Table 1: *A raw UDP Rotomotion Packet*

Offset	Field name	Type	Description
0	tv_sec	uint32_t	Time in seconds since the epoch
4	tv_usec	uint32_t	Fraction of a second
8	type	uint32_t	Message id
12	data	—	Message data

Table 2: *A DESIRED POSITION Message*

Name	Col	Offset	Type	Description
n	3	0	double	North position in m in the local tangent plane.
e	4	8	double	East position in m in the local tangent plane.
d	5	16	double	Down position in m in the local tangent plane.
hdg	6	24	double	Absolute heading in radians relative to north.

measures in case of communication loss. Typically, UAVs with on-board waypoint following will fly to and either orbit or hover at a 'loss of communication' position if the radio link is lost more than a specific time interval.

The short range and overhead of TCP/UDP make these protocols less popular than proprietary serial protocols for UAV applications. However, with the appearance of longer range ethernet (e.g. WiMAX), integrated chipsets, and the relatively sophisticated and standardized management of node addressing, it is inevitable that some derivative of TCP/UDP will be adopted eventually.

3.5 Rotomotion's UDP Message details

All messages have a 32 bit value that identify the type of message. In general, all measurements are in the local tangent plane, in meters or radians and referenced to a standard right hand rule coordinate system. Floating point values are IEEE doubles stored in the Intel representation. Integer values are store in network byte ordering. A packet consists of a 12 byte header followed by up to 65524 data bytes. Table 1 describes the raw UDP packet (network byte order) structure.

The payload may be extracted from the UDP packet by substracting the UDP 12 byte header from the total packet length and be interpreting the remaining content as appropriate for the message type. Each message type has a unique 32 bit message ID identifying it to the system (though only the lower 8 bits are currently in use). The most common messages are the `AFCS_STATE, AFCS_STATE, AUTOPILOT_STATE, FADEC, GPS_STATE, PPM, FLYTO, PLATFORM` and `TELEOP_CMD`.

An example of the `DESIRED_POSITION` message appears in Table 2. The desired position is sent as a response to a flyto packet. The UAV will try to fly to the position indicated in the NED frame. The message size is 32 bytes long. Table 3 lists the current message ids.

Table 3: *Though this small list belies the Rotomotion system's immaturity, these messages are sufficient for typical line-of-sight, experimental operations.*

ID	Enumeration	ID	Enumeration
0	COMMAND_NOP	23	VELSCALE
1	COMMAND_OPEN	24	MAGCAL
2	COMMAND_ACK	25	FADEC
3	COMMAND_CLOSE	26	TRIM
4	SIM_QUIT	27	AUTOPILOT_FFF
5	SIM_RESET	28	KILL
6	SAVE_CONFIG	29	TAKEOFF_PHASE
7	GET_CONFIG	50	ATTITUTE_GAIN_ROLL
8	SET_CONFIG	51	ATTITUDE_GAIN_PITCH
9	COMMAND_SETRATE	52	ATTITUDE_GAIN_YAW
10	AHRS_STATE	53	VELOCITY_GAIN_X
11	PPM	54	VELOCITY_GAIN_Y
12	GPD_STATE	55	VELOCITY_GAIN_Z
13	SVINFO	56	GUIDANCE_GAIN_X
14	AUTOPILOT_STATE	57	GUIDANCE_GAIN_Y
15	AFCS_STATE	58	GUIDANCE_GAIN_Z
16	FLYTO	80	SERVO_ROLL
17	TANGENT_PLANE	81	SERVO_PITCH
18	COMMAND_MODE	82	SERVO_YAW
19	PLATFORM	83	SERVO_COLL
20	DESIRED_POSITION	84	SERVO_ANTENNA
21	TELEOP_CMD	90	SIM_DISPLACE
22	FAILSAFE	255	NOTES

3.6 Visualization

3.6.1 X-Plane

The X-Plane flight simulator by Laminar Research[3] runs on a variety of hardware platforms including iPhone, Linux, Mac and Windows-based PCs. X-Plane also includes a powerful editor to build and customize aircraft and scenery, to produce a complete flight simulation environment. X-Plane's plugin architecture [4] allows users to create modules extending software's base functionality.

Like other flight simulators, X-Plane is intended for aerodynamic simulation and has very limited rigid body dynamics support. This deficit makes realistic air and ground vehicle simulations difficult. Nevertheless, the Rotomotion driver and X-Plane simulator were used successfully with Rotomotion Hardware during trials in 2006 at DRDC Valcartier.

[3]www.x-plane.com

[4]as discussed on the Wiki site: http://www.xsquawkbox.net/xpsdk/phpwiki/

4 Discussion

Increasingly, low-level vehicle control systems are being packaged as discrete products. This product evolution reflects the demand for simple teleoperated or semi-automated vehicle controllers and a supply driven by dropping production costs, and the appearance of novel MEMS sensors. A secondary result of such commoditization is the growth in unmanned vehicle interface protocols. From the standpoint of an integrated military unmanned vehicle control structure, these market changes are crucial but expose the growing need for vehicle integration into a coherent UAS control structure. The most notable attempt to achieve uniformity in military robotics, the Joint Architecture for Unmanned systems (JAUS), has had a mixed reception by the development community in part due to the fairly narrow initial focus on ground vehicles and an implied prescribed design for of vehicle control systems. Given that unmanned systems development has yet to settle on any particular vehicle software architecture, JAUS' scope has shrunk to a basic control instruction set, dropping any pretense to a prescribed vehicle architecture. Despite its limited success, the JAUS standard fails to address vehicle integration into military networks, a key requirement for larger UAS adoption – a void filled by many commercial proprietary solutions.

The focus of this brief study has been an examination of issues surrounding commodity autopilot integration into a networked vehicle control system. The autopilots assembled for this study are representative of the two basic approaches to UAS communications, serial RF communications and wireless UDP communications, and are arguably the most sophisticated in their class. Significantly, the wireless UDP solution by Rotomotion, is clearly less mature than CloudCap's Piccolo autopilot in the depth and breadth of controllable parameters and vehicle control algorithms.

4.1 Autopilots

While one could not expect identical autopilot design or performance from two independent suppliers, the variation between these two designs has significant implications for future net-enabled UAV services. The Cloudcap autopilot is complex, with a large memory and an arbitrary number of flight plans. Industry wide this is an unusual approach. Many systems store a finite number of specific paths: e.g. the current path, the loss of comms path, an autoland path, etc.. These internal differences will have significant impact on an exhaustive set of UAV services and have, to some degree, been captured within STANAG 4586 a GCS/Autopilot communication standard.

4.2 Aircraft Simulator

Both systems exploit rudimentary aircraft simulation allowing users to learn and practice UAS operations. Both simulators are closed to external inspection but open to parameter modification. Though adequate for mission practice, dynamic maneuver or more complex dynamics (e.g. slung loads for rotorcraft) will require more accurate and extensible simulation systems. In the long run, 'standard' simulation tools such as MATLAB represent a more flexible, rigorous approach to vehicle simulation.

4.3 Ground Control Station

Though Cloudcap's ground control station remains substantially more informative and flexible than Rotomotions, neither completely fulfills the long term requirements of multiplatform multivehicle control. Both systems have some form of moving map display and full vehicle state reporting. Both GCS systems are capable of multivehicle control with Cloudcap using Piccolo ID designators and Rotomotion using IP addresses. Not surprisingly both systems are bound to both proprietary hardware and, to a lesser extent, closed protocols. Similarities aside, Cloudcaps system is both more informative and easier to use.

4.4 Communications Interface

The Piccolo communication API is substantially more complex than the Rotomotion autopilot – a clear indication that Cloudcap has more flight experience. However, Rotomotion's use of standard networking tools makes vehicle communications both clear and familiar to novices with some internet experience – in the long run a more extensible solution than the Cloudcap model. Further, Cloudcap's system is neither open nor portable. The Cloudcap system relies on an unknown handshake protocol between client and GCS on initialization. This unpublished protocol ensures the user is dependent on Cloudcap or third party software. Since Linux communication libraries often lag behind the Windows versions, code portability becomes difficult with the Cloudcap model.

4.5 Software Design

Both DRDC designs succeed in embedding commercial UAVs into the MIRO infrastructure. The Rotomotion MIRO driver embodies the preferred practice, with minimal dependencies on external code, a proven publish-subscribe model, and direct communication with the autopilot. In contrast, the Cloudcap controller represents a 'simple', but unconventional communication solution based on commercial libraries to establish an indirect link with the autopilot moderated by the GCS. Even though the Piccolo continuously publishes flight data to the GCS, the GCS uses a client-server model where the client polls the GCS server for Piccolo data. Thus the Cloudcap MIRO driver is caught between a polling dialogue with the GCS and the demands of MIRO's publish-subscribe system. The MIRO driver compensates for this deficit through repeated (but wasteful) polling of the Cloudcap server and the Push timer mechanism. By comparison, the Rotomotion driver has more complex packet parsing, but less complex control flow, pushing events as received directly from the Rotomotion autopilot. Though the Rotomotion driver lacks the Push Timer and is thus somewhat less flexible than the Piccolo MIRO driver, both systems provide effectively identical event based performance to external consumers.

Clearly, the Cloudcap MIRO driver's dependence on commercial libraries simplifies development, but complicates support and portability of the driver. This approach greatly speeds development and exposes a large API to an external user. However, the library and GCS together hides a complex, proprietary low level serial communications and networking model between GCS and aircraft. Cloudcap's GCS server unnecessarily mimics some of these protocols and thus complicates communications with the GCS. Since Rotomotion's autopilot is founded on traditional 802.11g networking, the API is reduced to familiar IP networking and exchanged bytestreams – a portable and transparent API.

4.6 Visualization

A persistent weakness in the MIRO infrastructure is a simple, stable and reliable visualization environment. Though many proprietary rendering solutions are available, open source solutions are preferable for ease of collaboration and modification. In general, the benefits of open source must be carefully weighed against the complexity and effort of custom software development and maintenance – particularly for animation software. Open Source projects have the advantage that they can be very responsive to technological change and can rapidly adopt new or emerging standards and/or methods. However, this same responsiveness can and does lead to design instability where tools and techniques are always changing. So while systems such as Gazebo (or the nominally HLA-compliant Delta3D) are adequate in the short term, the simple engineering effort of growing these open source systems into stable military systems could prove difficult and costly in the long term. It is noteworthy that some open source projects (such as Flightgear) are remarkably sophisticated and demonstrate industry leading capabilities and comparable robustness (to MS Flight Simulator or X-plane, for example).

5 Future Work

In general, future development of UAV MIRO components will remain complicated by the base transport layer decisions taken by manufacturers. Since proprietary serial protocols will likely dominate the market for some time to come, the development of a stronger library independent driver using a common IDL seems to be the best way forward. While the risks of supporting new serial protocols will be ever present, the advantages of a common UAV IDL architecture, ideally a subset of a more general vehicle IDL protocol might be worth pursuing. This strategy, combined with appropriate information hiding techniques within the driver will insulate the greater MIRO network until the transition to TCP or UDP based UAVs.

As difficult as the communication structures may be, a further difficulty in MIRO based control is the common representation of vehicle and/or GCS memory models. In addition to proprietary communication protocols, manufacturers have adopted a wide variety of internal memory models. Some include multiple paths and emergency procedures on board, some reserve these functions for the GCS and others have only one active path at any time and all adopt widely different means of representing these functions. At some point a standard model will have to be developed that reflects an idealized UAV mission control system that, in turn, can be mapped through appropriate drivers onto a specific system's GCS and/or autopilot. Without a consistent and generic capability driver, communicating with UAVs will remain a vehicle specific procedure.

References

[1] Michael Basler, Martin Spott, Stuart Buchanan, Jon Berndt, Bernhard Buckel, Cameron Moore, Curt Olson, Dave Perry, Michael Selig, and Darrell Walisser. The flightgear manual.

[2] G. Broten, D. Erickson, J. Giesbrecht, S. Monckton, and S. Verret. Engineering review of ancaeus/avatar an enabling technology for the autonomous land systems program? DRDC Suffield TR 2003-167, Defence R&D Canada – Suffield, December 2003.

[3] G. Broten, S. Monckton, J. Giesbrecht, and J. Collier. *Software Engineering for Experimental Robotics*, drdc suffield sl Towards Framework-Based UxV Software Systems, An Applied Research Perspective, page 34. Number 2005-227. Springer Tracts in Advanced Robotics, 2006.

[4] G. Broten, S. Monckton, J. Giesbrecht, S. Verret, J. Collier, and B. Digney. Towards distributed intelligence - a high level definition. DRDC Suffield TR 2004-287, Defence R&D Canada – Suffield, December 2004.

[5] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

[6] S. Monckton, J. Collier, J. Giesbrecht, G. Broten, D. Mackay, D. Erickson, I. Vincent, and S. Verret. Staged experiments in mobile vehicle autonomy ii: Autonomous land systems demonstration results. DRDC Suffield TR 2006-243, Defence R&D Canada – Suffield, December 2006.

[7] S. Monckton, B. Digney, G. Broten, and S. Penzes. Layered autonomous overwatch: the necessity and feasibility of multiple unmanned systems in combat support. In G. R. Gerhart, C.M. ShoeMaker, and D.M. Gage, editors, *SPIE Defense and Security Symposium Unmanned Systems Technology IX*, number 2007-132, April 2007.

[8] Taylor. Real-time shuttle bus tracking in google earth | google earth blog. http://www.gearthblog.com/blog/archives/2006/02/realtime_shuttl_1.html, 2006.

[9] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar. MIRO - middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, 18:493–497, June 2002.

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence R&D Canada – Suffield PO Box 4000, Medicine Hat, AB, Canada T1A 8K6	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED (NON-CONTROLLED GOODS) DMC A REVIEW: GCEC December 2013

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Incorporating Commercial Autopilots into DRDC's Middleware for Robotics (MIRO)

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Monckton, S.; Desgagnés, R.; Gagnon, E.; Lauzon, M.

5. DATE OF PUBLICATION (Month and year of publication of document.) December 2010	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 50	6b. NO. OF REFS (Total cited in document.) 9

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Suffield
PO Box 4000, Medicine Hat, AB, Canada T1A 8K6

9a. PROJECT NO. (The applicable research and development project number under which the document was written. Please specify whether project or grant.)	9b. GRANT OR CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Suffield TM 2010-178	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
(   ) Defence departments and defence contractors; further distribution only as approved
(   ) Defence departments and Canadian defence contractors; further distribution only as approved
(   ) Government departments and agencies; further distribution only as approved
(   ) Defence departments; further distribution only as approved
(   ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Defence R&D Canada (DRDC) is extending its considerable experience in teleoperated air, land and seaborne systems to the development of autonomous systems for the Canadian Forces. Not surprisingly commercial organizations have developed a variety of unique and proprietary autopilots for unmanned aircraft vehicle control. Typically each provider claims multi-vehicle control as part of their product capabilities, but the lack of common standards between manufacturers makes such capabilities functionally of little value. Though some emerging standards (such as JAUS) show promise, DRDC has recognized that proprietary controllers are unavoidable and have chosen to incorporate such systems into a larger multi-vehicle system based on CORBA, an open networking standard. This report describes the integration of a GCS/autopilot system into the MIRO environment.

Recherche et développement pour la défense Canada (RDDC) désire accroître sa vaste expérience dans le domaine des systèmes aériens, terrestres et maritimes téléguidés en développant des systèmes autonomes pour les forces canadiennes. Bien entendu, des organismes commerciaux ont développé une variété d'engins autopilotés uniques et exclusifs servant à commander des véhicules aériens sans pilote. De manière générale, chaque fournisseur déclare que leur produit peut contrôler plusieurs engins en même temps, mais le manque de normes communes pour les fabricants fait en sorte que ces produits ne valent pas grand-chose sur le plan fonctionnel. Bien que certaines normes émergentes (comme les JAUS) semblent prometteuses, RDDC reconnaît que les contrôleurs privés sont indispensables, et a donc choisi d'intégrer ces systèmes au système multi-engins basé sur CORBA, une norme ouverte sur les réseaux. Le présent rapport décrit l'intégration du SCS/autopilote dans l'environnement MIRO.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

UAVs, autonomous robotics, multivehicle systems, autonomous intelligent systems