Improvements to Internal Combustion and Heavy-Lift Helicopters for Ardupilot

Google Summer of Code Application

Sriram Sami

Introduction

Ardupilot is known for its ability to autonomously control a large variety of vehicle systems, with a large focus on multi-rotor aircraft due to their ease of use and popularity within the hobby community. However, for certain applications, helicopter-type airframes provide benefits that multi-rotors currently do not. Helicopters are more stable and efficient aircraft, and therefore for similarly-sized airframes, helicopters tend to outperform multirotors in payload capacity and flight time. This comes at the cost of much higher mechanical complexity; a helicopter needs multiple servos to control its swashplate and needs a large main rotor, whereas a quadcopter only needs to vary the speed of its individual motors connected to small propellers. Helicopters also have more energy packed into their main rotor as compared to a similarly-sized multi-rotor airframe, making them more dangerous to operate. Therefore, autonomous helicopters have their niche in more commercial and specialized fields, where payload capacity and flight time are crucial metrics for success.

My team and I currently work on building **autonomous gas-powered tandem-rotor helicopters** aimed at transporting medical vaccines and cargo around the rural and mountainous regions of Papua New Guinea, and use Ardupilot daily for the control systems of the aircraft. We use a tandem design to be able to more easily load cargo onto the aircraft, due to its higher longitudinal center-of-gravity tolerance. However, this capability is **very new**[1] to Ardupilot, having been added less than three weeks ago and has yet to be fully tested. I recently contributed[2] to this codebase, as during setup of my tandem heli, I required functionality that has not been implemented. **Over the next few months, in the Google Summer of Code, I hope to to add more functionality to improve support for this type of airframe, all airframes that use ICEs, and helicopters in general.**

[1] https://github.com/ArduPilot/ardupilot/commit/1ad5e1db4edef23091768d2c43eb3b6dd7e118f6
[2] https://github.com/ArduPilot/ardupilot/pull/5965

Goals

I propose to add five new features, with one stretch goal, to Ardupilot.

1. Internal Governor
2. Electronic Fuel Injection System Integration
3. Payload Sensor
4. Landing Speed Adjustment based on Payload Weight
5. Rotor Speed Adjustment based on Payload Weight
6. (Stretch goal) L1 Navigation Controller Integration

Internal Governor

Helicopters require the RPM of their main rotor to stay constant during flight to provide lift, as the rotor speed will naturally tend to slow when under load, reducing lift. Currently, Arducopter controls throttle output through a few methods:

1. Directly send the pilot's manually-set throttle input as the throttle output.
2. Set the throttle output at a particular LOW or HIGH value based on the input from (usually) a two-way switch on the pilot's controller.
3. Setting a *throttle curve,* where the throttle output is determined by the position of the collective: an open feedback control loop.

Due to the need for finer, closed-loop control of the RPM, many helicopters have a **governor** of some sort, an external controller that can change the throttle output based on the rotor RPM it detects in order to keep the RPM stable. However, this is an additional component to be bought, and it would be better if Ardupilot was able to provide this functionality itself.

Electronic Fuel Injection System Integration

One of the planned additions is the integration of data from an **electronic fuel injection system (EFI) into Ardupilot.** The tandem frame I use has a DLE170 gasoline engine powering the main rotor system, used instead of electric power sources due to a much higher energy density, increasing flight time. The particular EFI being used (Ecotrons UAV EFI)[3] outputs engine rpm, throttle position, fuel consumption rate, and many other useful

[3] http://www.ecotrons.com/products/uav-engine-efi/

metrics over a **CAN bus**. While the flight controller I use (Pixhawk 2) has a CAN bus port, this data cannot be logged and used currently. At a minimum, logging this data provide information about the **engine RPM** which can be used with the **internal governor** proposed above to maintain the main rotor RPM, all through Ardupilot.

Payload Sensor

Another goal for a payload-carrying system would be to monitor how much weight is on-board, and adjust certain flight characteristics accordingly. I propose to add a simple sensor library that can detect the output from a basic weight sensor like the SparkFun Load Sensor (50 kg)[4], and this will allow further libraries to build on the in-flight monitoring of this data.

Landing Speed Adjustment based on Payload Weight

Based on the payload weight detected by the system above, I can implement a system to limit the force of landing on the landing gear by adjusting the maximum descent speed during landing. This would help to extend the life of mechanical components by ensuring that they endure a consistently low impact across multiple landings.

Rotor Speed Adjustment based on Payload Weight

Different payload weights require different amounts of lift to keep the aircraft aloft. Usually, this is handled by the control loops in Arducopter that vary collective output to maintain height, but with input from a payload sensor, the rotor speed setpoint could be dynamically adjusted to increase or adjust the lift provided without affecting the collective too much. This may be necessary since for large changes in payload weight, the set rotor speed may provide too much or too little lift and the collective might be over- or under-sensitive as a result.

However, it's important to note that at certain RPMs, the magnitude of the vibrations experienced by the frame increases by a large amount. These "vibration modes" need to be taken into account when designing this system so as to not exceed the limits of the system and the recommended limits of Ardupilot. Another factor that can affect this is the airspeed of the copter.

[4] https://www.sparkfun.com/products/10245

(Stretch Goal) L1 Navigation Controller Integration

If I am ahead of time for the previous five tasks, I can attempt to port the L1 Navigation Controller over to Arducopter. The L1 Navigation Controller provides more accurate navigation control as opposed to the normal crosstrack and PID-based navigation controller, and has successfully been used in Arduplane and Rover.

Implementation

Internal Governor

Assuming that the heli's current RPM is provided and being actively updated in one of the RPM_State structures of the APM_RPM module (https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_RPM/AP_RPM.h):

Minimally, certain parameters need to be defined for the governor system:

Parameter	Reason
H_GOV_SETPOINT_RPM	Defines the desired RPM for the rotor
H_GOV_RPM_INSTANCE	Selects which RPM instance is used for the governor
H_GOV_ENABLE	Enables/disables governor usage

For the control code, changes have to be made in AP_MotorsHeli_RSC.cpp (https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_Motors/AP_MotorsHeli_RSC.cpp). Specifically the code for `void AP_MotorsHeli_RSC::output(RotorControlState state)` has to be changed, so that if `state` is `ROTOR_CONTROL_MODE_CLOSED_LOOP_POWER_OUTPUT,` we attempt to close in on the RPM setpoint.

There already exists some RPM governing code[5] written for an older version of Arducopter by Robert Lefebvre that I plan to base this feature on.

Finally, a MAVLink message can be added to transmit the state of the governor state back to ground control stations, something like a GOVERNOR_STATE message that has 3 integer fields, governor_setpoint, current_rpm, and rpm_error.

[5] https://github.com/R-Lefebvre/ardupilot/commits/Copter-3.3.2-RPM-WIP3

Electronic Fuel Injection System Integration

The Ecotrons UAV EFI System provides sensor information over the CAN bus, using the UAVCAN protocol. Ardupilot already supports UAVCAN, so this integration is made much easier. First, parameters should be defined similar to:

Parameter	Reason
EFI_ENABLE	Enables/disables EFI logging for the Ecotrons EFI

Next, a library to read the EFI data should be created under **AP_EFI**, where a method should be scheduled to update the state of the EFI from the CAN bus at a particular frequency, something like `AP_Gripper_EPM::update_gripper()` (https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_Gripper/AP_Gripper_EPM.cpp)

The EFI library should provide a method that an AP_RPM backend can call to update its current RPM state. To support this, an additional enum entry needs to be added under `RPM_Type` in AP_RPM, followed by a new backend to support reading RPM from the EFI library. (https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_RPM/AP_RPM.h)

Finally, a MAVLink message can be added for ground station updates about the EFI state, like EFI_STATE, with fields like engine_rpm, throttle_position, fuel_consumption_rate, and other EFI state values.

Payload Sensor, Landing Speed and Rotor Speed Adjustment

The payload sensor should be implemented as a library like **AP_PayloadSensor,** with different backends defined for the various possible payload sensors. For the landing speed, a simple scale factor can be defined for the reading that will adjust the LAND_SPEED parameter in flight. This can similarly be implemented to adjust the rotor speed in flight, except that it needs to access vibration and airspeed data to make its decisions. Some of the parameters that can be added to support this are:

Parameter	Reason
PAYLOADSENSOR_ENABLE	Enables/disables the payload sensor
PAYLOADSENSOR_TYPE	Specifies the payload sensor backend to be used
PAYLOADSENSOR_LAND_SPEED_ENABLE	Enables/disables the payload sensor's adjustment of the landing speed
PAYLOADSENSOR_LAND_SPEED_SCALER	Based on the payload weight detected, decrease the landing speed by this multiplier of the payload weight
PAYLOADSENSOR_RPM_ADJUSTMENT_ENABLE	Enables/disables the payload sensor's adjustment of the RPM setpoint
PAYLOADSENSOR_RPM_ADJUSTMENT_SCALER	Based on the payload weight detected, increase the rotor RPM setpoint by this multiplier of the payload weight

Timeline (May 30th to August 29th)

I. Phase 1: May 30th to June 16th (EFI Integration)

This phase attempts to gather all the data from the EFI, log it to Dataflash logs, change the RPM backend to use the EFI data, and send back the data as a Mavlink message.

A. Phase 1.1: Connect to CAN bus of EFI and read data into Ardupilot.
    1. Deadline: June 4th
B. Phase 1.2: Log all data into dataflash logs
    1. Deadline: June 9th
C. Phase 1.3: Change RPM backend to use EFI data
    1. Deadline: June 13th
D. Phase 1.4: Send data back through MAVLink
    1. Deadline: June 15th
E. Phase 1.5: Make pull request to Ardupilot
    1. Deadline: June 16th

II. Phase 2: June 17th to July 10th (Internal Governor)

This phase implements the internal governor

A. Phase 2.1: Create parameters and write simple governor code in AP_MotorsHeliRSC
    1. Deadline: June 20th
B. Phase 2.2: Do bench-top tests with governor enabled on heli frames
    1. Deadline: June 28th
C. Phase 2.3: Conduct flight tests with various heli frames to check response time and accuracy of governor
    1. Deadline: July 9th
D. Phase 2.4: Make pull request to Ardupilot

III. Phase 3: July 11th to August 29th (Payload sensor)

A. Phase 3.1: Make hardware connection to payload sensor and attempt to read in raw data
   1. Deadline: July 15th

B. Phase 3.2: Convert payload raw data to weight info and confirm against a reference scale with multiple weights
   1. Deadline: July 22th

C. Phase 3.3: Conduct flight tests with various payloads and check if the flight data accuracy reflects the weight loaded onto the payload bay
   1. Deadline: August 1st

D. Phase 3.4: Write library to adjust rotor speed based on payload weight and scaler. Perform bench-top tests with a handheld RPM sensor to confirm the RPM outputs are as expected
   1. Deadline: August 8th

E. Phase 3.5: Conduct flight tests with different payloads and observe collective vs RPM vs amperes-drawn graph
   1. Deadline: August 15

F. Phase 3.6: Write library to adjust landing speed based on payload weight and flight test
   1. Deadline: August 20

G. Phase 3.7: Conduct flight tests of adjustable landing speed based on weight
   1. Deadline: August 27th

H. Phase 3.8: Cleanup and make pull request to Ardupilot codebase
   1. Deadline: August 29th

About Me

I'm a second-year computing student at the National University of Singapore (NUS), with a love of aviation, engineering and computing. My previous work with autonomous systems has been at HOPE Technik (http://hopetechnik.com/), where I built a custom ground control station for monitoring and controlling a swarm of autonomous octa-copters and autonomous landing pads for each copter for a choreographed light show during Singapore's National Day Parade 2016. All these systems ran Ardupilot, and I was inspired by the ability to modify the codebase to suit my needs, such as adding new MAVLink messages. I currently do part-time work at Yonah, a social enterprise run by undergraduates that is dedicated to empowering people in some of the most remote and inaccessible parts of the world using unmanned aerial systems. I also did two research attachments over three years at the A*STAR Institute for Infocomm Research, investigating financial crimes using social networks, and building sensor-based tele-physiotherapy systems. I also build systems for my residential college in NUS (Tembusu College). Some of my projects may be found at: https://github.com/frizensami/.

I would love to work on the Ardupilot project as it marries my love of aviation with programming, and being able to contribute to the vast community of open-source drone software users. I believe that my previous experience will allow me to contribute to the project productively over the period of GSoC, and hopefully in the future as well.