A HIGHLY EXPANDABLE LOW-COST OPEN-SOURCE UAV SYSTEM WITH HIGH ON-BOARD PROCESSING POWER

Marcin Kmiecik[*] , Krzysztof Sibilski[*] , Wieslaw Wroblewski[*]
[*]Wroclaw University of Technology, Poland
marcin.kmiecik@pwr.wroc.pl; krzysztof.sibilski@pwr.wroc.pl; wieslaw.wroblewski@pwr.wroc.pl

Keywords: *UAV, autopilot, open-source, ROS, SLAM*

Abstract

The paper presents a novel open-source UAV autopilot architecture, currently supporting multirotor Vertical Take-Off and Landing (VTOL) type aircrafts. It is mainly aimed at research purposes. It utilizes an ArduPilot Mega (APM) [1] autopilot hardware, a PM-PV-D5251 Single Board Computer (SBC) [2] for high on-board processing power and Robot Operating System (ROS) [3] software for the ease of development and code reusability. The system's capabilities have been presented by performing an indoor scene mapping by means of an open-source SLAM algorithm's implementation.

1 Introduction

Over the past 10 years, the Unmanned Aerial Vehicle (UAV) market has grown rapidly and it is expected that this market expansion will continue for the foreseeable future. While much of this growth is attributed to defense applications, there are an increasing number of applications for UAVs in the commercial and university research sector. This is particularly so for smaller sized UAVs categorized as Miniature UAVs. While the quadrotors' popularity has only increased in recent years, there are many existing designs available. These designs can be broken down into two main categories: toys and professional commercial products. While the first are not suitable for research because of eg. very low payload, short flight time or non-modular hardware and/or software design, the latter's main disadvantage is very high cost. Platforms available currently for academic research known to the authors in terms of costs also fall into the last category. Prices ranging from 10000USD still make it beyond the reach of most research units in developing countries not sponsored by defense agencies or governments. In our work we have focused to fill this gap and provide a below 1000USD solution with good performance and flight time for rapid development.

Our design originates from legacy ArduCopter code. It has completely rewritten using only the driver (low-level) portion of the aforementioned project (as a reference driver implementation). It is a layered, object-oriented architecture currently intended for multirotor VTOL UAV's. It provides a set of application programming interfaces (APIs) together forming the main layers, that UAV designers usually have to implement, allowing developers mostly to concentrate on functionality. Furthermore, it provides a Hardware Abstraction Layer (HAL), for easy on-board device changeability and integration of new drivers. The main benefit of the project is the already mentioned ROS compatibility. ROS provides libraries and tools to help software developers create robot applications. It's fully supported on Linux, but other experimental versions for different operating systems are also available, including Windows and OS X. ROS is licensed under an open source, BSD license. It provides hardware abstraction, device drivers, libraries, visualization tools, messagepassing, package management, and more.

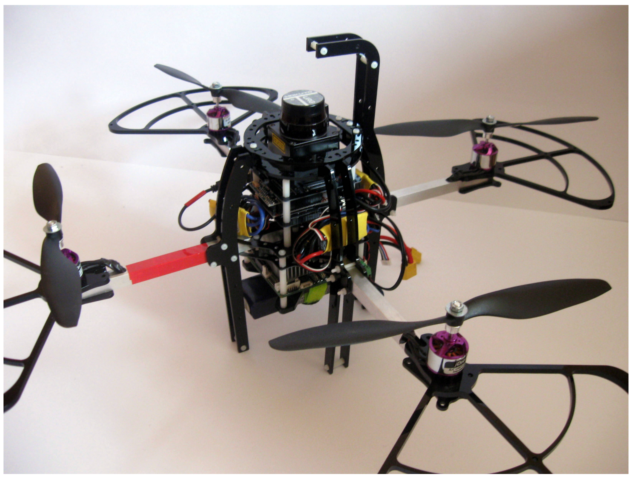

Fig. 1 Quadrotor platform created and used in the lab

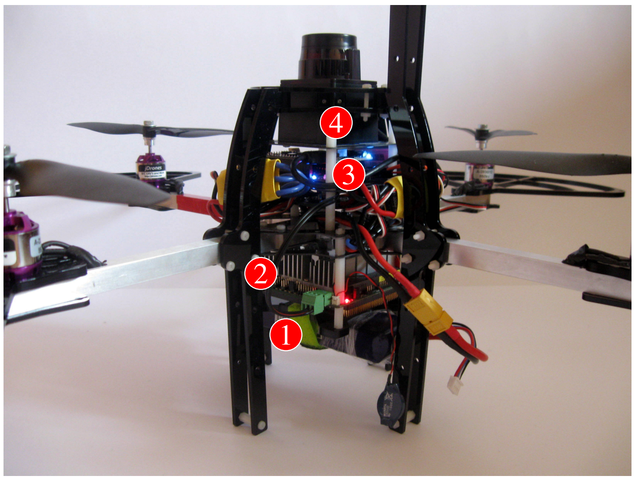

Fig. 2 Quadrotor platform test-bench for stabilization strategies testing. 1) LiPo battery, 2) PM-PV-D5251 SBC, 3) ArduPilot Mega, 4) Hokuyo URG-04LX-UG01 Laser Scanner

A complete, ready to fly platform weights slightly above 1kg with a 3300mAh LiPo battery included, providing 600g of extra thrust for additional payload. The quadrotor is capable of 15 minutes hover without additional payload. In our four-rotor configuration, its diameter measures 70cm and it is 30cm high.

2 Hardware Description

The project is based on ArduPilot Mega (APM) autopilot created by the DIY Drones Internet community. APM hosts an Atmega2560 Atmel AVR processor running at 16MHz with 256k of flash memory. APM is used to interface Inertial Measurement Unit (IMU) and other onboard sensors, perform attitude estimation and drive up to eight motors. An onboard PM-PV-D5251 computer featuring a dual core Intel Atom 1,8GHz processor, 2GB RAM and four USB 2.0 ports (see [2] for further details) provides additional computational power for high-level tasks. Both computers interchange data over an RS232 to USB FTDI chip. Connectivity with a ground station is ensured via IEEE 802.11b/g USB wireless network card added on-board (but any other radio module exploiting available hardware interfaces can be used). See Fig. 3 for a block diagram.

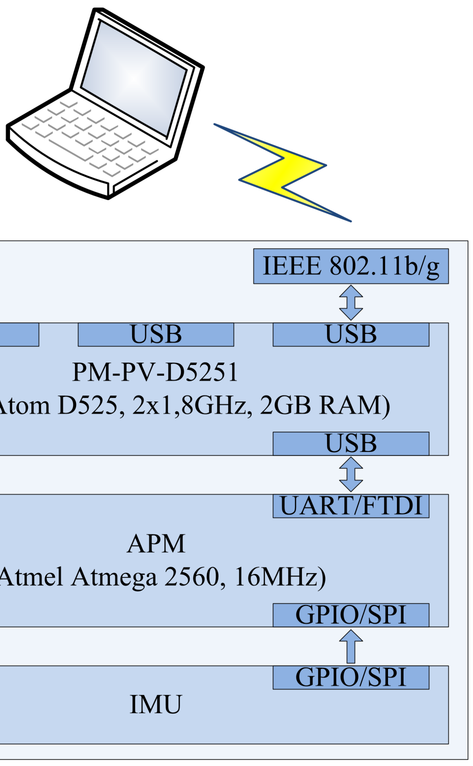

Fig. 3 Hardware architecture

3 Software Architecture

The system's software architecture has been designed with the below priorities in mind:

- ROS compatibility, hence the choice of C++ programming language.
- modularity at a degree allowing for system's components to be easily maintained or replaced, without the risk of causing neighbouring modules' failures or unconsciously altering their behaviour — this directly led to utilizing a strictly object-oriented design.
- least possible dependency of microprocessor's specific libraries, hence effort has been made to move all hardware dependent code to a Hardware Abstraction Layer (HAL).

All the above requirements led to a design shown on Fig. 4. All components pictured will be de-

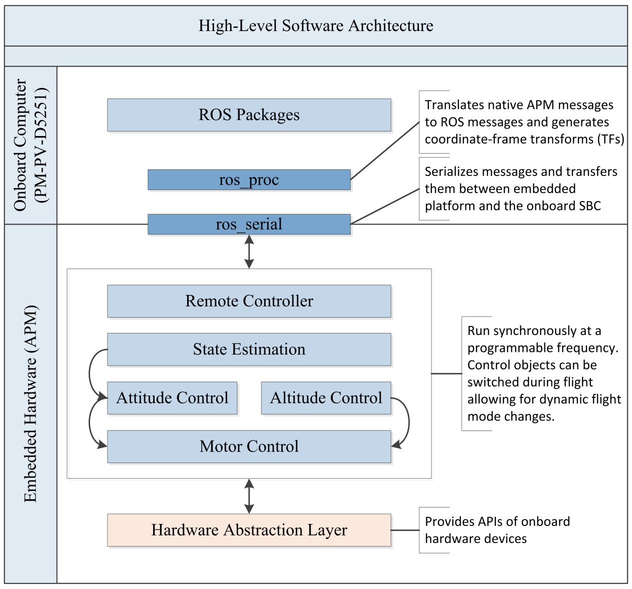
High-Level Software Architecture
Onboard Computer (PM-PV-D5251)
ROS Packages
ros_proc
ros_serial
Translates native APM messages to ROS messages and generates coordinate-frame transforms (TFs)
Serializes messages and transfers them between embedded platform and the onboard SBC
Embedded Hardware (APM)
Remote Controller
State Estimation
Attitude Control
Altitude Control
Motor Control
Run synchronously at a programmable frequency. Control objects can be switched during flight allowing for dynamic flight mode changes.
Hardware Abstraction Layer
Provides APIs of onboard hardware devices

Fig. 4 High-Level Architectural Design

scribed in more detail in later sections.

The core, i.e. embedded part of the system, runs in a synchronous fashion, with the only exception of microprocessor interrupts for the sake of I/O operations and time-based work-scheduling. Each of the control classes must implement an `IController` interface, a specialization of which is available for each kind of controller. This allows for in-flight controller replacement in turn making dynamic flight-mode changes possible (e.g. `NullAltitudeController` can be replaced by `SonarAltitudeController` allowing for turning altitude-hold on basing on user's input). An exemplary controller hierarchy is shown on Fig. 5.

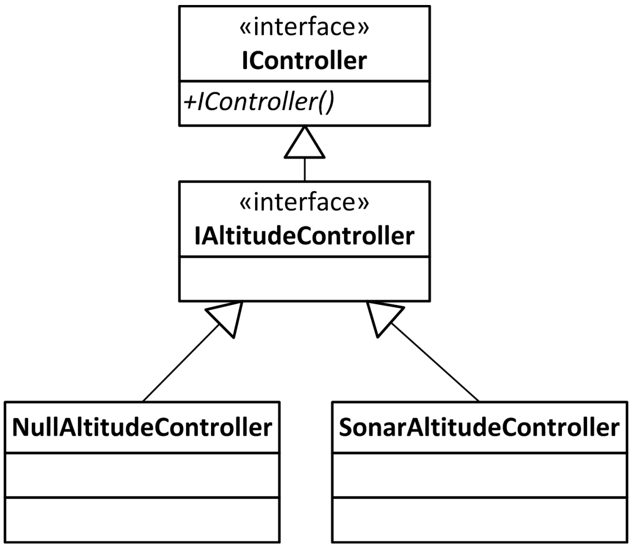

Fig. 5 Controller class hierarchy

3.1 Remote Controller

The `RemoteController` class is responsible for timely reading user's input (in a manual UAV operation mode). It then stores current readings in a structure passed further to other components, which makes them independent of `RemoteController`'s implementation. As the spoken module has access to ROS data, it can easily be reimplemented to either work basing on ROS-provided manual input, hence eliminating the need to use expensive RC-equipment, or to fully automate UAV's operation.

3.2 State Observer

A separate class-hierarchy is provided for state estimation. The reference implementation uses a complementary filter as described in [4]. It is a robust and fast attitude estimation algorithm well suited for the low-power ATmega1280 microprocessor board available onboard. Yaw drift cancellation is made possible with a Hokuyo URG 04LX laser scanner measurement provided via ROS drivers. The complementary filter has been directly defined on a $SO(3)$ rotation matrix, i.e. the structure of the filter used and its error formulation build on R defined in $SO(3)$.

3.3 Attitude Control

Attitude controller constantly updates percentual thrust corrections to be sent to motor controller. For that sake it reads `StateObserver`'s estimated attitude and attitude-rates and provides input for `MotorController`. It's programmed using PD control assuming decoupled horizontal X-Y dynamics, with an inner loop for pitch- / yaw-rate damping, as suggested in [5].

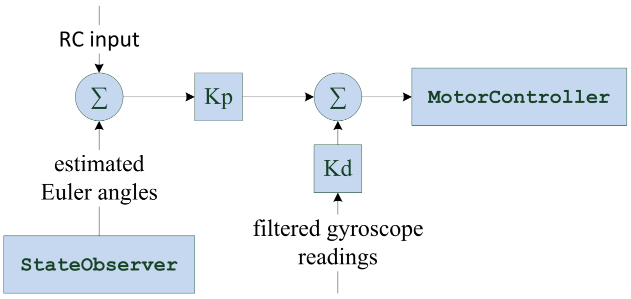

Fig. 6 Horizontal PD-controller's structure

3.4 Altitude Control

The `AltitudeController` component implements a PID-loop for SONAR-based altitude control, linearized around hover-thrust (recognized as the moment altitude control mode has been turned on). When activated, it provides initial thrust for `MotorController`.

3.5 Motor Control

The task of motor controller component is to send motor commands according to desired total thrust and percentual thrust corrections for angular orientation provided by attitude controller. The `IMotorController` class'es API is independent of the amount of propellers, hence providing an abstract interface for all kinds of VTOL aircrafts propeller systems. A reference implementation is provided for a quadrotor aircraft.

3.6 Hardware Abstraction Layer

HAL's function is to hide possible differences of various hardware specifics. In case the system ought to be ported to another hardware platform, most of the higher-level code wouldn't have to be altered (it is very hard to assure no changes would have to be made in case of a shift from an AVR μc to e.g. ARM architecture). It comprises of a set of pure virtual classes (interfaces), hence enforcing functionality each type of hardware component has to provide, see Fig. 7 for an example. Access to all drivers is made possible via an abstract factory, one of which have to be implemented for each hardware platform — a factory implementation returns proper `ISonar` (singleton) object for the current platform, an `APMSonar` in our case.

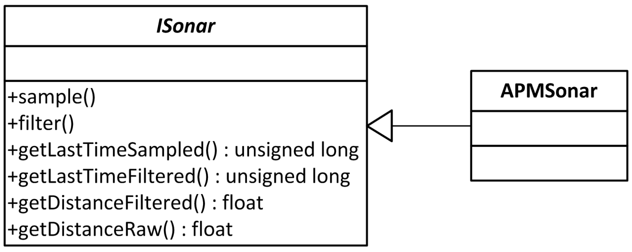
ISonar
+sample()
+filter()
+getLastTimeSampled() : unsigned long
+getLastTimeFiltered() : unsigned long
+getDistanceFiltered() : float
+getDistanceRaw() : float
APMSonar

Fig. 7 Application Programming Interface of a SONAR sensor

3.7 ROS Compatibility and Logging

For the sake of data interchange between ROS system and the autopilot, a serialization mechanism had to be provided. A decision was made to utilize ROS Serial package [6], which is a point-to-point version of ROS communications over serial, primarily for integrating low-cost microcontrollers (Arduino) into ROS. ROS serial consists of a general p2p protocol, libraries for use with Arduino, and nodes for the PC/Tablet side (currently in both Python and Java). An additionally provided `ros_proc` package generates coordinate-frame transforms and publishes them as ROS TFs. This enables for full two-way communication, data-logging and presentation using ROS-provided means.

The autopilot gives broad logging possibilities. Each component (specifically `IController`) can be configured or re-programmed, to log data as ROS-topic at a desired frequency. These informations can be stored using `rosbag`, a ROS program designed for data recording and playback. Another application, `rxbag`, can be used for data visualization and generating charts (see section 4 for an example). Finally, thanks to a simple, yet effective mechanism for launching distributed systems over SSH called `roslaunch`, it is possible to run the whole solution with just one terminal-command.

4 Example of Use

We've chosen to present capabilities of our plaftorm by implementing a process called Simultaneous Localisation and Mapping (SLAM). It enables autonomous mobile robots to localize in previously unexplored environments and incrementally construct a map of its surroundings. Only open-source, freely available ROS packages have been used.

For indoor odometry, a laser scan-matcher [7] package has been used. It is an incremental laser scan registration tool allowing for scan matching between consecutive laser scanner messages, and publishing the estimated position of the sensor. The algorithm was fed with additional roll-pitch data from the autopilot's IMU to provide alpha-beta filtering [8] and serves as an odometry estimator. It utilizes Andrea Censi's Canonical Scan Matcher [9] implementation.

We have used GMapping [10, 11, 12] as a SLAM solution. GMapping is a highly efficient Rao-Blackwellized particle filer to learn grid maps from laser range data for which a ROS wrapper is available.

Lastly, `hector_map_server` and `hector_geotiff` packages of `hector_slam` [13] ROS-stack have been used for travelled path and obtained map visualization.

It has been chosen to log data from `StateObserver` component and GMapping package. Some possible charts are shown on Fig. 8, 9, 10 and 11 below.

5 Summary

Integrating an Open-Source autopilot with ROS thus makes it possible to reuse cutting edge algorithms developed by the community for UAV control and navigation, data logging, online autopilot status monitoring, multi-agent cooperation, etc. Its very low price makes is available for small research institutes, or even hobbyists.

All hardware (rotors, frame, propellers) can be picked arbitrarily, depending on design requirements. The hardware-design freedom further decreases maintenance costs, as crashes are an unevitable cost-factor in the process of new UAV algorithms development, and it's usually best to pick building parts easiest available on the local market for fast replacement.

During design, another high-power ARM-based SBC (BeagleBoard-xM) have also been tested. We have succeeded to compile and install ROS and integrate it with the autopilot. Though it has shown to have too little computational power for running SLAM algorithms, which led to PM-PV-D5251, it still may provide enough for many users and hence should be considered if lower cost is a priority.

A noticeable drawback of current autopilot board (APM) is the lack of native USB support of ATmega2560 which limits APM to ROS communication bandwith — which we hope will be solved with future APM releases.

The project source-code and documentation can be found under http://code.google.com/p/4clover/.

References

[1] Anderson, Chris et al., *ArduPilot Mega*. http://code.google.com/p/ardupilot-mega/.

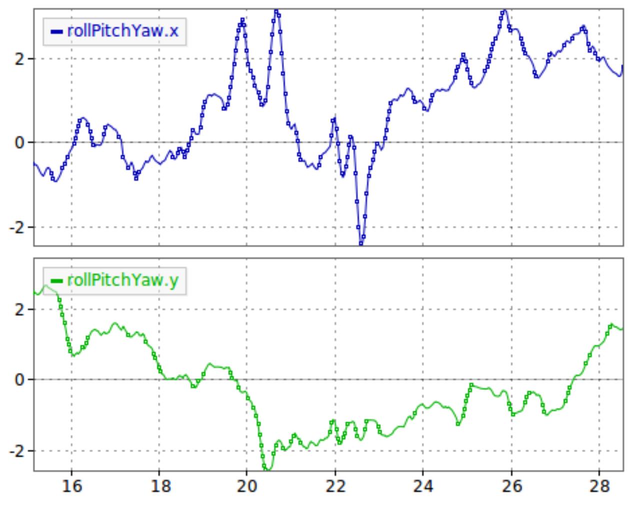

Fig. 8 Estimated roll and pitch angles (in degrees)

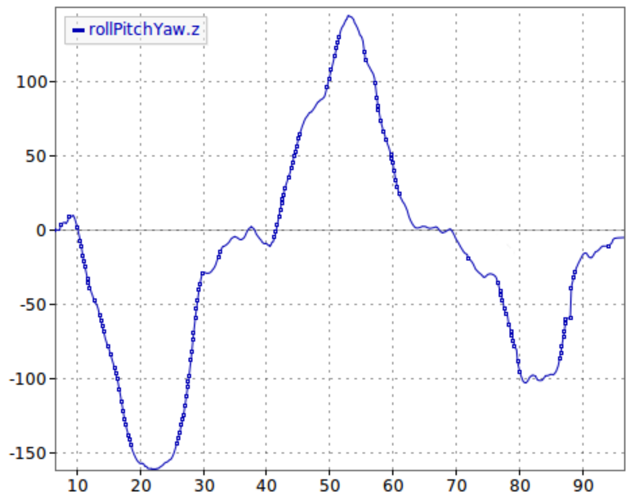

Fig. 9 Drift corrected yaw angle (in degrees)

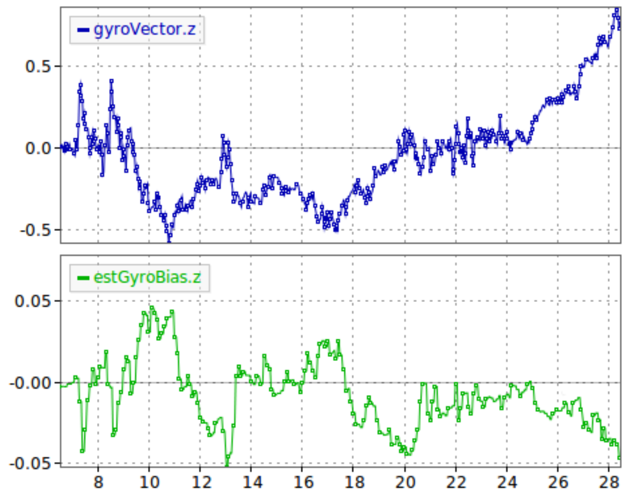

Fig. 10 Raw gyroscope yaw-rate reading vs measured estimation error (in deg / s)

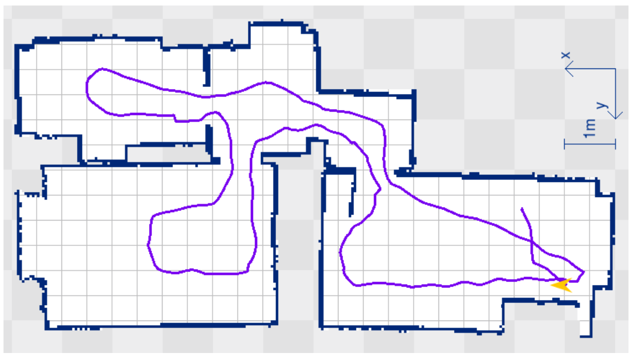

Fig. 11 SLAM algorithm outcome

[2] iEi Technology Corp, *PM-PV-D5251-R10 Technical note*. http://www.ieiworld.com/.

[3] Quigley, Morgan et al., *ROS: An Open-Source Robot Operating System*, ICRA Workshop on Open Source Software, 2009.

[4] Mahony, Robert et al., *Nonlinear complementary filters on the special orthogonal group*, Automatic Control, IEEE Transactions on, 53(5):1203-1218, June 2008.

[5] Beard, Randal, *Quadrotor Dynamics and Control*, Brigham Young University, February 2008.

[6] Stambler, Adam, Ferguson, Michael, *ROS Serial* http://www.ros.org/wiki/rosserial.

[7] Dryanovski, Ivan, Morris, William, *Laser Scan-Matcher* http://www.ros.org/wiki/laser_scan_matcher.

[8] *Alpha-Beta Filter* http://en.wikipedia.org/wiki/Alpha_beta_filter.

[9] Censi, Adrea, *An ICP variant using a point-to-line metric* Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2008.

[10] *GMapping* http://openslam.org/gmapping.html.

[11] Grisetti, Giorgio et al., *Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters* IEEE Transactions on Robotics, 2006.

[12] Grisetti, Giorgio et al., *Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling* Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2005.

[13] *Hector SLAM* http://www.ros.org/wiki/hector_slam.

Acknowledgement

This work has been supported by a grant sponsored by Polish National Centre for Research and Development for years 2010-2013.

Copyright Statement