Project: Drone Labs

Project manager: Tomáš PLch (*tomas.plch@gmail.com*), KSVI

Team:

Vojtěch Sedláček
Martin Skalský
Oto Petřík
Tomáš Dzurenko
Robert Husák
Lukáš Krížik

Date of completion: June 2014

Motivation

Robot prototyping and development is a difficult task not only to enthusiasts, but to professionals as well. The design and construction of a robot consists of three major phases

a) hardware design

b) construction design and realization

c) software development.

In respect to hardware, there is the notion of utilizing templated open source hardware kits that are housing the necessary hardware - i.e. processing units, controllers, sensor kits etc. It is also possible to develop custom hardware for a specific task or robot, but it is expensive and tedious task. While there is some reusable control software, generally the options are limited and the development is hard and complicated due to lack of development environments and debugging tools. Debugging a live robot can be a hard and dangerous task. The introduction of aerial robots has made all the above tasks even more complicated and expensive. Given the current state, we decided to create a development environment for drone control software.

Inspiration

Despite the abundance of general integrated development environments such as *Visual Studio*, *Eclipse* or *Netbeans*, we believe that development of drone control software should be easier. Conceptually, we are inspired by the *Pogamut* project developed at the faculty for the last 8 years and its idea of reducing programmer's workload by providing matched platform and development environment. Encouraged by component systems in embedded domain (e.g. *ProCom*), we believe that it is viable to base the system on distributed components.

Problems

In order to develop drone control software, developer has to tie together different parts of hardware and software. Various drones and input devices have different APIs and support libraries, often in multiple incompatible versions. Developing and especially debugging program on such platform is an unpleasant experience. Complexity of the problem lies in rather large range of abstraction levels (from low level input APIs to limited high level drone libraries), limited operating system support and issues caused by opaque internals of most drones. As such the field can be considered hostile to newcomer developers. Developing control software for multiple drones or running on multiple nodes can be daunting at best.

Goals

Our primary goal is to simplify development of control software for aerial robots, namely the *R-UAV* (*Rotor Unmanned Aerial Vehicles*), also known as quadcopters. The main trait of the application will be the capability to create control software and artificial intelligence mechanisms for a custom *R-UAV*. The application GUI and the underlying middleware will allow user to create components and simple programs to control the *R-UAV*. The component middleware will allow simpler porting between robotic platforms. We also aim at providing simple debugging features for tracking and inspecting the robot's status and operation.

Project structure

Our proposed solution consists of multiple loosely connected parts. We expect most of the development to be done in parallel, with the exception of the initial stage of the component system implementation.

Component-based environment

In order to allow development of reusable drone control software, we will provide a component system implemented in C++. Initial inspiration came from *ProCom - the Progress Component Model*. However, we will implement a component system ourselves. Where deemed useful, we adopt design features of *ProCom*, but we do not aim to create strict (or full) reimplementation of *ProCom*.

Our component system will consist of two separate layers. Lower layer is used to create tightly-coupled, low-level component structures with none to very low communication overhead. Higher layer is used to connect high-level components, possibly running in different processes, using messages sent through established connections between components' endpoints. User will have the option to implement higher-layer component by instantiating a lower-layer component system and use small amount of C++ code to pass (and optionally transform) messages between the layers.

Some experimental work on the component system has been already done. For the lower layer, the very first prototype based on class generation using templates was already implemented. For all intents and purposes, design of the higher layer is completed, main 3rd party libraries for its implementation were selected (*Protocol Buffers*, *ZeroMQ*, *Boost::ASIO*) and some experimental code was written.

ProCom reference:
`http://www.idt.mdh.se/pride/data/documents/ProComRefManual-v1.1.pdf`

Scripting

As a demonstration of usability of our component-based environment, we'll also implement simple support for scripting. Scripts can be used to control robots in a similar way as they are used in computer games to manipulate with NPCs and their behaviour. Loader and interpreter will be represented as a component or set of components keeping other parts of system relatively independent. Scripts can be used to create more complex tasks for robots without detailed knowledge of underlying language.

IDE

In order to fully take advantage of resulting platform, additional support tools are required. To make developer experience as easy as possible, we will provide an augmented IDE. It will be implemented as modification to *QtCreator* (either as a plugin or custom build), to avoid implementing an IDE from the ground up.

Drone pilot application

As a *"pilot"* application to show benefits of our architecture we will create *Drone Pilot*. Its purpose will be to enable piloting drones using several types of controlling devices. Those comprise not only joystick and gamepad, but also *3DConnexion Space Navigator*, sometimes called 3D mouse, and *Novint Falcon*, 3D positioning device with force feedback. The application will utilize the component system in order to maximize its extensibility and maintainability. Potential developers will be able to simply create bindings with types of drones and controlling devices not originally available in the application. While we expect users to develop their applications conveniently using our IDE, the development of the pilot application will begin before the IDE is complete, as they will be developed in parallel.

Test vehicle

Based on our experience, we chose *AR.Drone* and *Bitcraze Crazyflie 10DOF* quadcopters as testing vehicles. While there are open source libraries for communication with these drones, we take advantage of rather simple communication protocols and we will implement our own software to handle drone communication and connection to component system for better integration with our project structure and its core facilities. The other reason not to use open source libraries is their relatively low quality and high maintenance cost. We would need to enhance their functionality to support our own more generic protocol for communication with the rest of the component system. We expect that maintaining forked low-quality libraries would be harder than writing simple protocol library ourselves.