UNIVERSITY OF VICTORIA, COMPUTER SCIENCE DEPARTMENT

CSC-586A Self-Adaptive Systems

Tory Borsboom, Sean Debroni, Stephan Heinemann

May 25, 2015

1 PART III – SENSOR APIS

1.1 APPLE IOS GYROSCOPE API

1.2 ANDROID MAGNETIC COMPASS API

1.3 NUTTX / ARDUPILOT PX4 OPTICAL FLOW AND SONAR API

The *ArduPilot*[1] autopilot stack supports a wide range of sensors to be used with any of the supported vehicle types including *Plane, Copter* and *Rover*. It builds on top of the *NuttX*[2] operating system and requires the applicable firmware for the underlying hardware such as the *Pixhawk*[3]. The supported sensor types include those to capture data about the vehicle's *environment* as well as those related to its own *capabilities* in order to establish an adequate *situational awareness*.

Examples of supported environment sensors gather *inertial* (attitude and acceleration), *barometric* (altitude and airspeed), *magnetic* (direction), *range* (distance), *optical* (orientation and recognition) and *satellite* (position) data. Sensors that capture the vehicle's capability state include a *battery monitor* (power consumption and endurance) and a *performance monitor* (scheduler latency). Environment sensors themselves may provide *health* and *quality* monitoring fuctions. Furthermore, all information that is provided by the *NuttX* operating system about its *Pixhawk* platform including the available *memory* and *filesystem* space can be instrumented to enhance the vehicle's capability state.

[1] http://ardupilot.com/
[2] http://nuttx.org/
[3] https://pixhawk.org

The ArduPilot Application Programming Interface (API) separates sensor *frontends* (interfaces) from their actual *backends* (implementation) in order to easily support different sensors of the same type. These and other Hardware Abstraction Layer (HAL) modules can be found in the `libraries` directory[4] and are, hence, separate from any concrete vehicle-related software. The *Copter* vehicle employs the sensor API through its `sensors.pde` arduino sketch file which represents an even higher level API[5] to the actual vehicle functions such as different autoflight modes. As a representative example, the structure of the *PX*4 *Optical Flow / Sonar*[6] sensor interface shall be explained. This sensor allows to capture the optical flow, that is, the rate of translational and rotational changes in a captured image, in order to establish positional information. The optical flow camera is mounted on the bottom of the *Iris*[7] quadcopter body pointing downwards and sampling the floor pattern that is being overflown. Since the aircraft can be tilted (pitch and bank) and flown at different altitudes, a range finder (sonar sensor) on the same sensor board is employed to relate the optical flow information to the measured ground distances accordingly. Assuming the actual attitude of the aircraft based on inertial sensors, an actual ground distance can be derived from the sonar slant range readings.
Figure 1.1 shows a small extract of the optical flow sensor frontend API in the `libraries/AP_OpticalFlow` directory possessing the sensor state data structure as well as methods that determine flow and body rates in a two-dimensional image. The interface itself furthermore consists of methods that determine the health of the sensor and the quality of the data being captured by it.
Figure 1.2 shows an API extract of the range finder component of the optical flow sensor board. On the *PX*4 *Optical Flow* board, a sonar sensor is employed to obtain range in order to relate optical flow to ground distance. This concrete implementation is abstracted by the more generic range finder API in the directory.
The `sensors.pde` arduino sketch file uses the above API in order to provide the *Copter* vehicle with necessary sensor data to perform its functions such as automatic flight modes. The overall static structure is shown in Figure 1.3.

[4] https://github.com/diydrones/ardupilot
[5] http://www.arduino.cc/en/pmwiki.php?n=Reference/HomePage
[6] https://pixhawk.org/modules/px4flow
[7] http://3drobotics.com/iris/

```
class OpticalFlow
{
    //...
public:
        //...
    bool enabled() const { return _enabled; }
    bool healthy() const { return backend != NULL && _flags.healthy; }
    void update(void);
    uint8_t quality() const { return _state.surface_quality; }
    const Vector2f& flowRate() const { return _state.flowRate; }
    const Vector2f& bodyRate() const { return _state.bodyRate; }
    uint8_t device_id() const { return _state.device_id; }
    uint32_t last_update() const { return _last_update_ms; }

    struct OpticalFlow_state {
        uint8_t device_id;
        uint8_t  surface_quality;
        Vector2f flowRate;
        Vector2f bodyRate;
    };
private:
        //...
};
```

Figure 1.1: Optical Flow Sensor API

```
class RangeFinder
{
public:
        //...
    struct RangeFinder_State {
        uint8_t                         instance;
        uint16_t                        distance_cm;
        uint16_t                        voltage_mv;
        enum RangeFinder_Status         status;
        uint8_t                         range_valid_count;
                //...
    };
    //...
    void update(void);
    uint16_t distance_cm() //...
    int16_t ground_clearance_cm() //...
    RangeFinder_Status status(void) //...
    bool has_data() //...
    uint8_t range_valid_count() //...
private:
        //...
};
```

Figure 1.2: Range Finder API

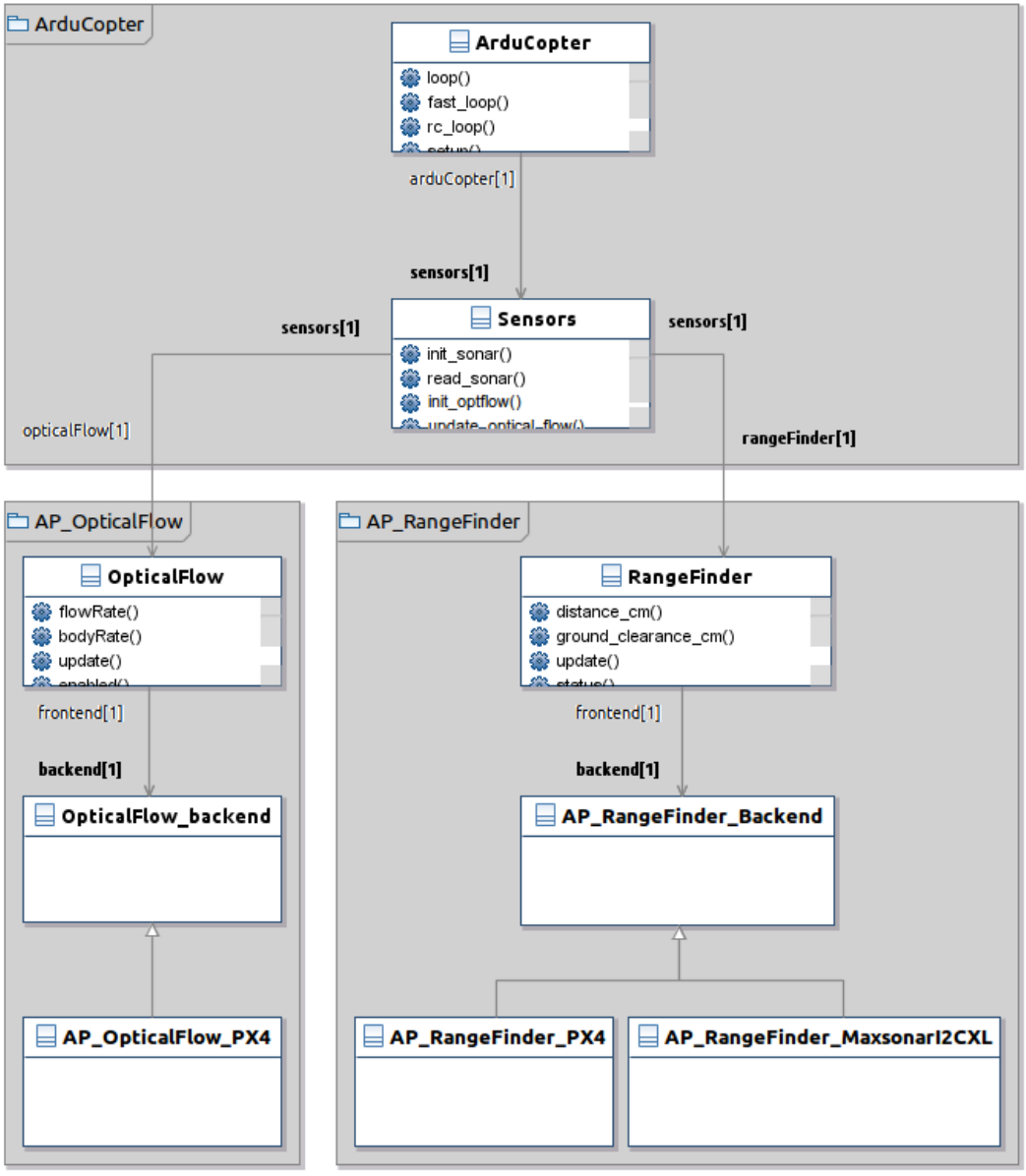

Figure 1.3: PX4 Optical Flow Sensor Board API

2 ACRONYMS

API Application Programming Interface

In computer programming, an application programming interface is a set of routines, protocols, and tools for building software applications. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types. An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising each other. [8]

CPS Cyber-Physical System

A cyber-physical system is a system of collaborating computational elements controlling physical entities. Today, a precursor generation of cyber-physical systems can be found in areas as diverse as aerospace, automotive, chemical processes, civil infrastructure, energy, healthcare, manufacturing, transportation, entertainment, and consumer appliances. [9]

HAL Hardware Abstraction Layer

Hardware abstractions are sets of routines in software that emulate some platform-specific details, giving programs direct access to the hardware resources. [10]

II Industrial Internet

The Industrial Internet refers to the integration of complex physical machinery with networked sensors and software. [11]

IIoT Industrial Internet of Things

The Industrial Internet of Things is a subdiscipline of the IoT, which describes IP-enabled systems in factories, offices and other commercial (and sometimes government) facilities. It is the part of the IoT that focuses on how smart machines, networked sensors and sensor analytics can help improve business-to-business initiatives across a wide variety of industries, especially manufacturing. [12]

IoT Internet of Things

The Internet of Things is the network of physical objects or "things" embedded with electronics, software, sensors and connectivity to enable it to achieve greater value and service by exchanging data with the manufacturer, operator and/or other connected devices. [13]

SoSs System of Systems

System of systems is a collection of task-oriented or dedicated systems that pool their

[8] http://en.wikipedia.org/wiki/Application_programming_interface
[9] http://en.wikipedia.org/wiki/Cyber-physical_system
[10] http://en.wikipedia.org/wiki/Hardware_abstraction
[11] http://en.wikipedia.org/wiki/Industrial_Internet
[12] http://itlaw.wikia.com/wiki/Industrial_Internet_of_Things
[13] http://en.wikipedia.org/wiki/Internet_of_Things

resources and capabilities together to create a new, more complex system which offers more functionality and performance than simply the sum of the constituent systems. [14]

ToE Theory of Everything

A theory of everything or final theory, ultimate theory, or master theory is a hypothetical single, all-encompassing, coherent theoretical framework of physics that fully explains and links together all physical aspects of the universe. [15]

ULSS Ulta-Large-Scale System

Ultra-large-scale system is a term used in fields including Computer Science, Software Engineering and Systems Engineering to refer to software intensive systems with unprecedented amounts of hardware, lines of source code, numbers of users, and volumes of data. [16]

[14] http://en.wikipedia.org/wiki/System_of_systems
[15] http://en.wikipedia.org/wiki/Theory_of_everything
[16] http://en.wikipedia.org/wiki/Ultra-large-scale_systems

References