Software System Architecture Document

# Project ADSEN86 – Automated Drone Sentry

## List of Changes to Document:

**Draft Version: 1.0**

Initial version: 11/23/2024

Minimum Viable Product version: 1.0

### Project Name: Automated Drone Sentry

### Project ID: ADSEN86

## Introduction:

This document provides the overall Software system architecture of the ADSEN86 project. ADSEN is Automatic Drone Sentry that has AI capabilities to follow its main user’s commands and perform sentry function. This architecture document provides a high level overview of the entire system and how the drone is architected to provide its service.

## Abbreviations:

**CFI:** Core Function Intelligence

**DRIM:** Display Renderer and indication Module

**CMI:** Connectivity Module Interface

**MFM:** Motor Function Module

**VIM:** Vision Interpreter Module

**SM:** Sensor Module

# Design Diagram

### Basic Framework

CORE

FUNCTION

INTELLIGENCE

ADSEN86

Motor Function Module

ADSEN86-001

Vision Interpreter Module

ADSEN86-002

Connectivity Module Interface ADSEN86-005

Display Render and Indication Module ADSEN86-003/004

Sensor Module

ADSEN86-001

Control Signals

Data Signals

Figure 1

The diagram represented by Figure 1 provides a basic framework that the Sentry drone is designed around.

**Core Function Intelligence:**

The core functionality of the drone is operated from the Core Function Intelligence Module. This module controls the operations of the drone and the decision making of the drone based on inputs from different modules. The 5 main modules are implemented as per the Basic Requirements Documentation. These modules feed into the Core Intelligence and also operate based on information provided by the core intelligence module.

**Motor Function Module:**

This module provides the functionality of movement to the Drone sentry. This module mainly takes the input from the Core intelligence module.

**Vision Interpreter Module:**

This module provides the functionality of gathering the camera based input data. Sending the input for interpretation to the AI sub functionality and determining result which can be used to make decisions.

**Connectivity Module Interface:**

This module provides the ability to connect to the drone sentry over multiple different interfaces. The type of interface to connect over can be selected using the settings

**Display Renderer and Indication:**

This module provides the ability to indicate to the User the intentions of the Drone and the results of what it is seeing and interpreting. This can be through different LED based signage or Display system.

**Sensor Module:**

This module provides the inputs from different sensors to generate information of the drone’s surroundings. This input can be used by the CIM to figure out which way to move and obstacles to avoid.

# Connectivity Module Interface

***Introduction:***

The connectivity module interface provides a basic framework that exposes outward communicating hardware modules and technologies from the drone subsystem. The Connectivity Module has 3 different layers. Each layer abstracts a portion of the underlying interface and exposes less details to the higher up services.

The basic framework of the Connectivity module is provide an interface to the top level service. See the image below for more details

***Service Layer***

Service 4

Service 3

Service 2

Service 1

Protocol 1

Protocol 2

***Protocol Layer***

Protocol 2

Protocol 6

Protocol 3

Protocol 4

Protocol 5

Protocol 1

***Interface Layer***

The connectivity layer will consist of 3 layers. The Protocol Layer, Interface layer and the Service layer.

**The Service layers** provide a specific service that needs to be enabled.

For example let us say that we need the ability to configure the drone using Bluetooth. We need to provide some form of a command line interface that can set certain Wi-Fi settings or other settings in the drone. A Bluetooth command line service can run that acts as a shell. It will respond to commands and provide specific feedback that a certain setting was configured.

Another example is let us say we need the ability to output certain specific strings or video data over a Wi-Fi interface that is encoded and then sent over an encrypted link. The process of figuring out what video needs to be sent and where to fetch that data is done by the service layer.

This layer is generally designed as a factory.

**The Protocol layer** is a series of decorators that provide cascading levels of functionality that satisfy a specific protocol. The Standard interface protocol is the base protocol that will have the specific interface that will communicate over the LOB. All other protocols will use a lower level protocol as the lower layer of communication and are instantiated/called from higher level protocols or services.

In our example of the Bluetooth command line, the protocol layer may consist of the standard protocol that has the Bluetooth interface. Then it may be wrapped by a HID or SPP profile protocol layer that talks in terms of the Attribute table and instantiates the Bluetooth to talk in that format.

Similarly the protocol layer in the Wi-Fi video stream example may be the standard Wi-Fi protocol Interface wrapped by the TLS1.3 protocol wrapped by the HTTPS protocol interface.

This layer is designed as a Decorator.

**The Interface layer** is the final layer that uniquely specifies a hardware technology. It has all the functionalities that setup the hardware technology.

In the case of the Bluetooth command line the Bluetooth Standard protocol may communicate with the Bluetooth Low Energy Interface that sets up the Bluetooth capability. It provides APIs to setup the GAP and GATT table for the same.

In the case of the Wi-Fi video stream the Wi-Fi standard protocol talks to the Wi-Fi interface class and configures the Wi-Fi to be able associate and provide a socket connection as server or a client.

# Vision/Audio Interpreter Module

**Introduction:**

This module is responsible for capturing an image from the camera, capturing audio from an audio source and then using this information to perform an inference using the Neural processing unit to obtain an inference based on which a decision can be made to perform some action using other services.

The basic layout for this is as shown below

Service 2

Service 3

Service 4

***Service Layer***

Service 1

Inference Engine

Models

Capture Subsystem

Media DRIC

GStreamer library and service

Tensor Flow Delegate and library

***Kernel Space***

V4L2 Subsystem

Audio Driver

NPU Driver

Camera Driver

I2S

USB Driver

**Media DRIC:**

Media DRIC or MANDRIC is an acronym for Media Display Render Infer and Capture. The MANDRIC is a module that supports the ability to capture video or Audio, send the media to an inference engine to make decision, display the video to a specific interface and do 3D graphics processing generate rendered output. The MANDRIC is a support layer that fits in to the GStreamer platform to allow for smooth communication between the 4 operations. At the higher layer it support the basic functionalities and provides a way to connect the different operations.

*Capture Subsystem*: The MANDRIC will consist of capture subsystem that will facilitate the capture of streaming video from the camera.

**Inference Engine:**

Inference Engine is a module that supports the ability to load Machine Learning Models and Neural Nets into it and it will facilitate the processing of the Machine learning models and the neural nets to perform the inference. The Inference Engine behaves like the gun and the models are the bullets that perform the Neural Inference. The Models are switchable and can be switched between execute. The goal is to make this switching very easy to do and hence very efficient.

**Models:**

The models are objects that will contain the detailed Tensor Flow graph models of the specific model that needs to be loaded into the inference engine. It will have additional meta data that may be needed for that specific model like pre-processing and post-processing functionality that the model will nedd to produce useful information from what is seen.

**GStreamer Layer:**

GStreamer is a 3rd party layer that provides ability to perform multiple operations. To support Neural Net activity the GStreamer app will support the NNStreamer Plugin that gives the ability to run Tensor Flow models on the device through the Neural Processor. Gstreamer makes the ability to talk between the capture, inference, display and the OpenGL based 3D rendering extremely easy and supports a Sink and Source based design that is very easily plug and play.

# Display Renderer and Indication

***Introduction:***

This module is mainly responsible for enabling the display, the 3D rendering of graphics and rendering design for the LED module. These three aspects of this module are called separately by different services that perform operations on the video buffer.

The basic design layout is as follows

***Service Layer***

Service 4

Service 3

Service 2

Service 1

LED Renderer

Display Renderer

Illustrator Slate

Illustrator Slate

Illustrator Slate

Illustrator Slate

Vision Interpreter Module

3D Renderer

Media DRIC

V3D Driver

I2C Driver

Display Driver

hoverLED driver

***User Space***

***Kernel Space***

DRM Driver

KMS Driver

GStreamer library and service

**Illustrator Slate:**

The Illustrator slate is a context that is present within each service that will allow the service to access the Display renderer, 3D Renderer and the LED Renderer. This slate tries to make the rendering of a design agnostic to the slate it is drawing to. For example we can have a common design that needs to be rendered to the LED and the Display screen and the Illustrator slate will possess the basic designs that can be rendered. However the individual representation of each of these designs will be understood by the underlying renderer modules.

**Display Renderer:**

The display renderer is a subset of the display functionality that may use Weston or some library like GStreamer that will render on a display. The subset functionality is the functionality that will provide the buffer only that needs to be displayed. It won’t have all the bells and whistles of what needs to be done for displaying using Weston. The Display renderer also acts as a conduit for the 3D renderer. This conduit will take the graphics buffer and combine it with the display buffer to produce 2 windows to display the content on.

**3D Renderer:**

This is truly one of the main renderers that will generate a design that needs to be displayed. While Display renderer acts like a conduit, the 3D renderer is what generates the design using the graphics libraries and OpenGL compliant code that will render the shape and design. The 3D renderer will talk to the Mesa libraries and also further down to the Display Rendering Manager in the Linux kernel. It will use the 3D graphics hardware to render the shapes that is needed to be displayed.

**LED renderer**

The hover craft will have a small LED matrix display that will support a certain small resolution. Several designs may need to be rendered on this display. The logic to render the different designs will be provided as a matrix that will be illuminated in the RGB format. To generate the RGB values and the movement of the different LEDS the LED renderer will generate the matrix. It is important that this module is configurable to different LED sizes and hence it will have a hardware specific layer at the Linux kernel that will configure the appropriate hardware. The output from the LED render can be made standard.

# Motor Control Module

**Introduction:**

This module is responsible for controlling the motor subsystem of the hover craft. The basic input to this module is the general rate per unit vector for distance travelled and also the vector that determines a spherical radial point from the hover-crafts current position. See the diagram below for more information. The different bands represent the distance a unit vector will travel.

z

x

y

The Motor control module is entirely in the kernel module. The direct interaction with the motor allows the driver to drive the relevant GPIO and simply translate the vector to a motor speed for the different motors. The speed will ensure that the hover craft moves quickly to that position. See the break down below.

Service

***Kernel Space***

Motor Control Module

GPIO/Motor Driver

Motor Control SHIM

**Motor Control Shim:**

The Motor control SHIM is the main interface that will take the input of the rate of movement and the vector length. The SHIM will then provide a way to send this information to the kernel module that controls the motor subsystems. This layer simply acts as a conduit for the motor to be controlled.

**Motor Control Module:**

The motor control module is a Linux kernel module that will take in as inputs the unit distance vector change rate and then use the vector provided to move the hover craft to that particular location. The module will contain two parts. The first part is the part the will convert the vector to individual motor control values. Second part will contain the adjusted mapping and calibration that will provide the exact movement. The module will also have a way to understand which hardware is being used and adjust the speed and torque of the motor appropriately.

The module will essentially take the vector values and then using the reference direction will convert them into specific motor rotation values that will provide movement of the hover craft towards that direction.

**GPIO/Motor Driver**

The GPIO and Motor driver is a general description for the module that will drive the appropriate PWM and GPIOs to make the motor rotate at a particular speed.

# Sensor Module

**Introduction:**

Sensor 1 The sensor module is responsible for collecting sensor information and then sending the information to whichever service requests the sensor data. Sensor data is generally time sensitive and in some cases soft real time. This makes it very important that collection is prioritized so that sensor data is received within certain fixed timeline. Generally multiple sensor information may need to be collected. So it is important that the sensor that has the highest rate of collection is prioritized. The best design approach is to make this a publish-subscribe model that multiple services can request off of a single sensor data or multiple combinations of sensor data. But at the lower level the sensor information is timestamped and collected in a queue that can be dispatched to whoever requested it.

Sensor N Module

Sensor 2 Module

Sensor 1 Module

***Kernel Space***

***Sensor Manager***

Sensor N SHIM

Sensor 2 SHIM

Sensor 1 SHIM

Sensor Manager

Thread

Sensor API Layer

***Context List***

Sensor

N

Sensor2

Sensor1

***Sensor Queue***

***Service Layer***

Service 4

Service 3

Service 2

Service 1

**Sensor API Layer:**

The sensor API layer is the interface that provides different services to register the sensor data as and when it comes. This API will allow the services to request a context, destroy a context. It will allow the services to request a subscription at a particular rate and will provide a way to expose a callback when the sensor event is triggered. This API will isolate all the other layers.

**Sensor Manager:**

The Sensor manager is a module that will manage multiple sensor data and handle the grabbing of the data from the underlying sensor. It will then queue this data in exclusive queue meant for each sensor and then parse and find which services will need to be notified about the event or should it wait. It basically will tag each Quantum of sensor information into a timestamp and wait for the timer to expire when it will send the data back.

**Sensor SHIM:**

The senor SHIM is a layer at the lower level of the Sensor manager that essentially act as interrupt driven or polling driven channels. As we get information from the Sensor the data is extracted from the SHIM and placed in the Queue. Internally the SHIM will use underlying OS primitives to communicate to the kernel modules that perform the hardware level calls.

**Sensor Kernel Module**

The sensor module is the actual kernel module that will perform the operation of handling the sensor hardware to elicit a response from it. Any calculation that needs to be done will be done by this module.