ALPIDE Visualiser

Bringing Cutting-Edge Monolithic Active Pixel Sensor Technology to Schools: the ALPIDE Arduino Shield

CERN ALICE

Matthew Aquilina

Supervisor: Magnus Mager (EP-AID-DT)

Co-Supervisor: Felix Reidt (EP-AID-DT)

June - August 2017

Contents

1 Synopsis

1.1 Background

ALICE (A Large Ion Collider Experiment) is one of the four main experiments present on the Large Hadron Collider (LHC) at CERN. Its main purpose is to study the physics of quark-gluon plasma, a phase of matter which is assumed to form through the collisions of heavy lead ions at high energies.

In 2020, the ALICE Inner Tracking System (ITS) will be upgraded with Monolithic Active Pixel Sensor (MAPS) particle detectors[1]. The ALPIDE chip is a custom-made MAPS, made specifically for instrumenting the ALICE detector.

1.2 Project Overview

The main purpose of this project was to interface the ALPIDE chip with a low-cost and accessible table-top system.

The system needs to be able to automatically interpret and display data in an easy-to-understand package, in a manner useful for pedagogical applications in high schools and universities. Of course this also means that this system needs to be simple and intuitive to install and setup.

2 Supporting ALPIDE Hardware

This section details the hardware and low-level microcontroller interface used to communicate with the ALPIDE chip[2].

2.1 ALPIDE Overview

The ALPIDE chip is a rectangular 15 mm by 30 mm detector containing 512 by 1024 pixels, along with a periphery circuit region. Each pixel cell contains several analogue components (in addition to the sensing diode) as well as several digital components including a multi-event buffer and a pixel masking register. Typically, a readout occurs by first 'triggering' each pixel (strobing the pixels and collecting hits if present) and then reading from the multi-event buffer all the available hit positions (binary result).

2.1.1 Interfacing

The chip has systems built in which allow it to interface with other electronic systems in a variety of ways. Apart from high speed serial and parallel data ports, more suitable for use with FPGAs or other high speed electronics, the chip has a bidirectional control line which is used to relay commands and write/read registers. However, through this single control line, one can also read out event data. This means that the ALPIDE can be completely controlled, triggered, and read via this single line (if a lower read-out speed is acceptable). A block diagram of the ALPIDE chip showing the different data transfer ports available is shown in Figure 1.

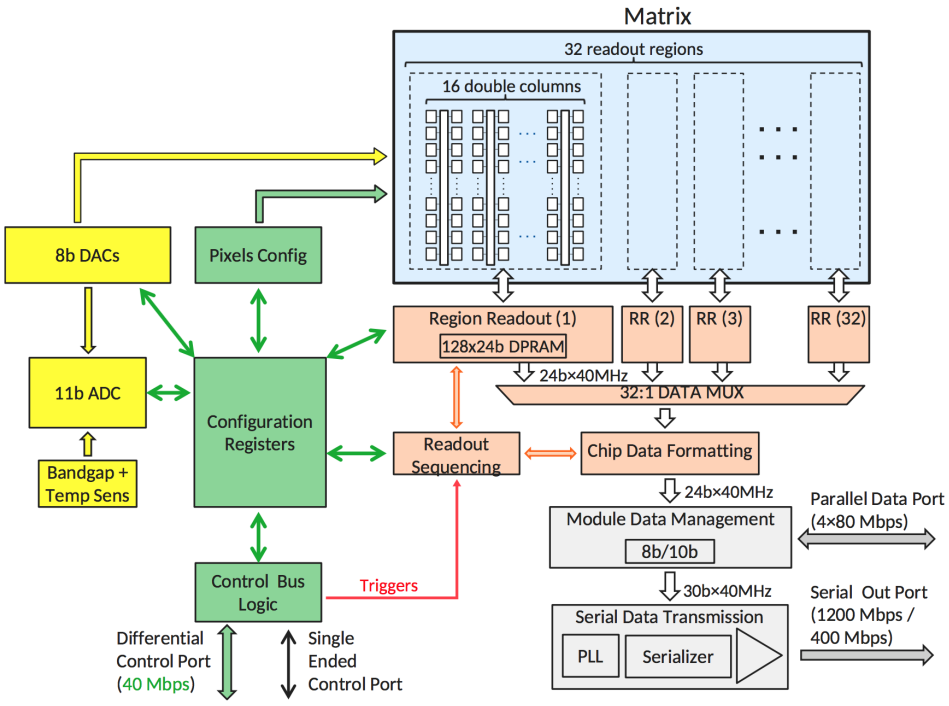

Figure 1: ALPIDE Block Diagram, taken from the ALPIDE Operations Manual[3]

Despite the PLL (and hence the serial link), ALPIDE's logic is fully static. Thus, it does not need a periodic clock of a certain frequency running continuously. A relatively cheap microcontroller can be used to control, trigger, and read out data from the chip all from the same line without sending out a regular clock. The clock used for transfer (driven by another line) is completely in the hands of the microcontroller. This feature is what makes the ALPIDE chip so easily interfaceable with non-complex, digital systems such as an Arduino microcontroller.

2.1.2 Protocol

The ALPIDE chip can be completely controlled from an Arduino development board; by sending out a clock signal and reading/writing data to the control port. Of course, the ALPIDE chip has a large amount of registers which control nearly every aspect of its functionality. One needs to write to specific registers to setup the system for use.

All this can be taken care of through the Arduino programming environment; once the protocol is set up, the system can repeatedly trigger the ALPIDE chip and read out data.

More details on the operation of the ALPIDE chip, its different modes, and available configurations can be found in the ALPIDE Operations Manual[3].

2.2 Arduino Shield

An Arduino Shield (PCB) was specifically manufactured to interface this chip with an Arduino UNO (or other Arduino boards with a similar PCB layout).

The shield design includes voltage regulators which convert the Arduino provided voltages

to the correct standard/level(3.3V for the LVDS transceivers and 1.8V for the ALPIDE chip), the means to manually set a plane ID (via dipswitch) for each shield, a temperature sensor and all the routing necessary for correct operation.

The design also allows stacking planes one over the other, with data/power pins shared through each plane. In order to build up this stacked system, functionalities such as unique chip IDs to address data/commands to particular chips on the shared control line are required. Some commands can also be broadcast to all planes on the stack simultaneously. A typical Arduino and ALPIDE stack is shown in Figure 2.

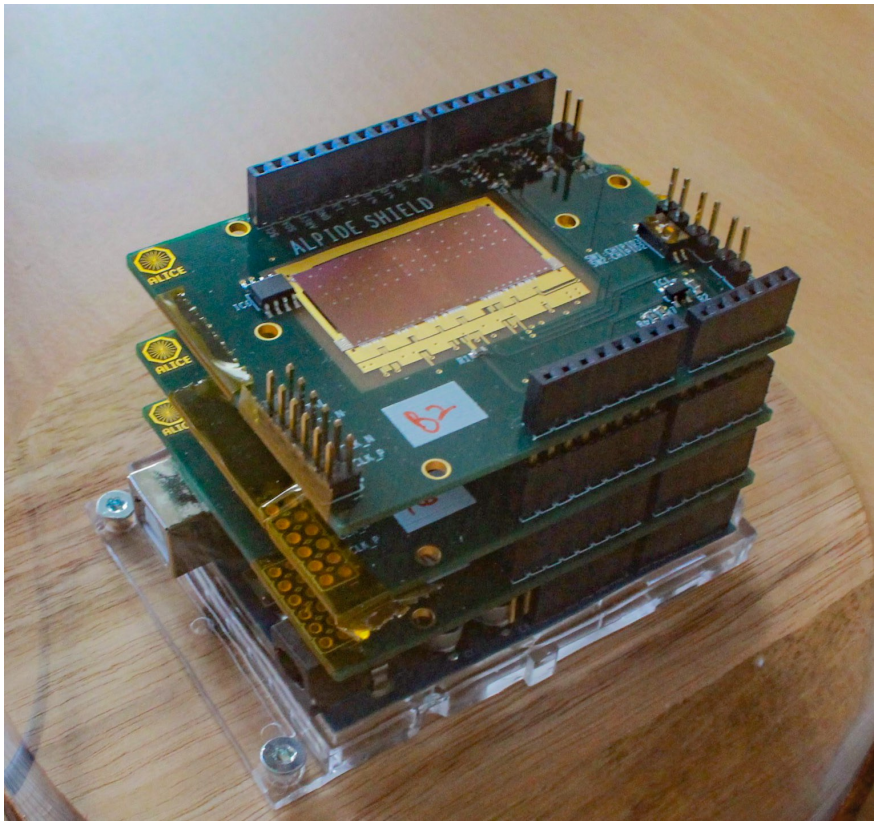

Figure 2: ALPIDE Stack of 3 Planes

The shields used in this project are the first version of the ALPIDE shield. A second version has been designed which frees a number of pins on the Arduino with specific functions and has a smaller length(fitting more comfortably on the Arduino board). These improvements and others make the second version superior to the first, but still has much the same functionality.

2.3 Stack Functionality

As stated, the data gathered from each ALPIDE shield in a stack will be the position of pixels which register a 'hit'. In a stack, this data can be used for a variety of purposes.

The most straightforward application is that with three or more planes stacked up, a hit on each plane can be used to reconstruct the trajectory of a particle passing through each sensor.

This can be tested using a radioactive source, however, the chips will also react to cosmic radiation (muons)[4]. Thus, a stack of ALPIDE chips can be used to reconstruct muon paths.

Calculations at this level, however, are best done on a faster computer. Ideally, the Arduino should act as the low level interface between the ALPIDE stack and a computer. In this project, data is sent to the computer via USB, however there are other options which could have been explored, such as the use of a Wi-Fi shield to broadcast data.

3 Bringing the ALPIDE Sensor to Life

3.1 Bridging the Gap

A low-level controller system is absolutely necessary for interfacing with the ALPIDE chip. However, this is not the easiest of systems to understand and work with. Its text-based interface can be cumbersome to work with. A GUI-based system is probably much better to make the system more user-friendly and accessible to non-technical people and students.

The software selected to bridge this gap was the Unity Game Engine[5].

Unity, a system normally used for creating cross-platform 3D games, offers the means to bridge the gap between the technical and visual world. This tool allows one to simultaneously receive data from the Arduino system, decode it, perform calculations and display the results whilst also providing an entire suite of tools for designing a whole 3D system.

The following subsections contains a description of the inner workings of the ALPIDE Visualiser that was built in Unity.

3.2 Implementation

3.2.1 System Overview

The diagram in Figure 3 provides a top-level overview of the system built in Unity.

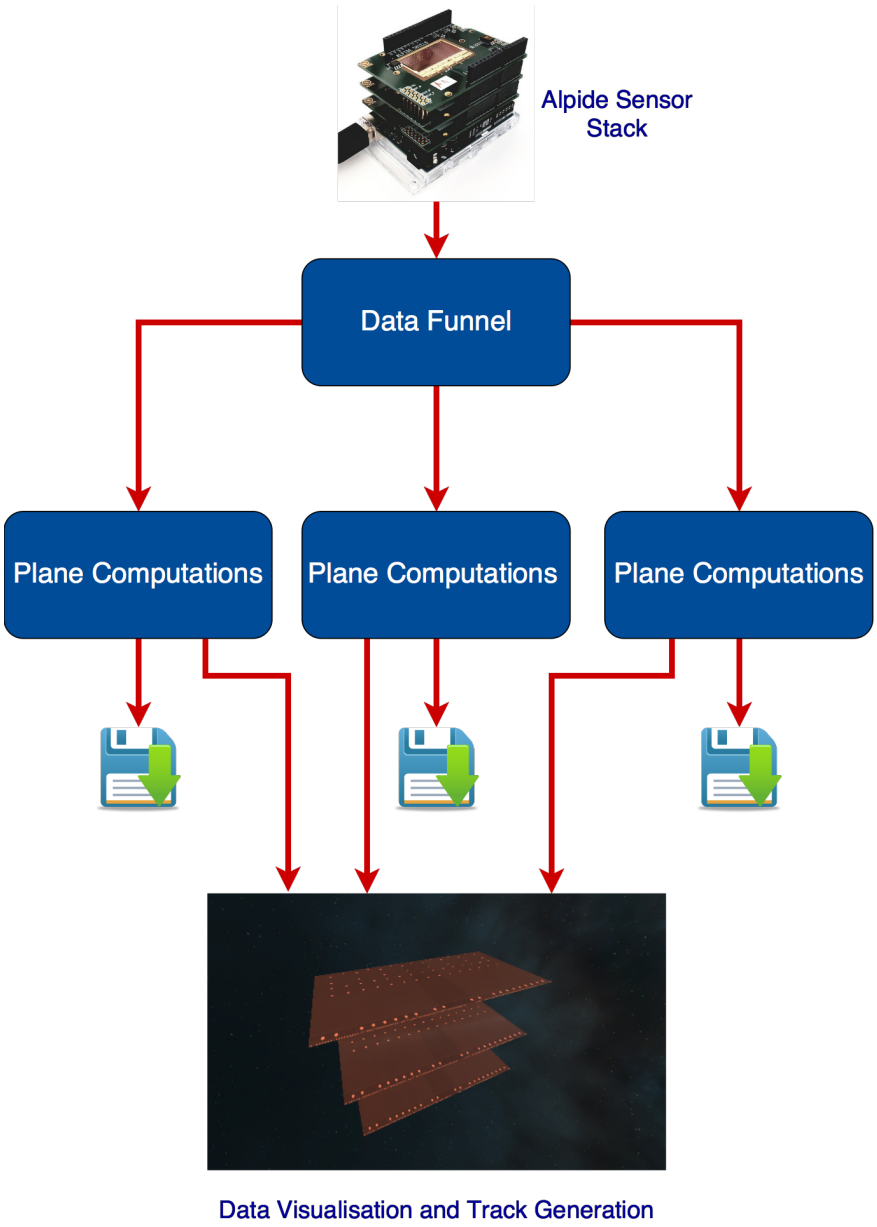

Figure 3: Top Level Block Diagram for Unity System

Data is sent from the Arduino to the Unity system, containing the hits for each plane on the stack. The data is split and sent to different plane scripts (for as many planes are placed in the stack). Here, each script interprets an individual plane's data. This interpretation is then either saved for further analysis, or used in deducing whether a muon track has been detected.

The following subsections contain a brief overview and some insight into the various parts of the mechanisms which make the system work.

3.2.2 Gathering Data From the Arduino

The Arduino itself has been set to send the data it gathers in JSON[6] format. The data is simply the x,y position of each hit registered by an ALPIDE chip in the last trigger period.

Each plane data is seperated by a special character, which can be used as a marker for splitting the input string between each virtual plane.

The system must be capable of reading out data at the same rate it is sent from the Arduino. In case of data being read incorrectly, checks have been set in place which validate the data read before passing it on to further stages.

3.2.3 Visualising Planes

In order to correctly represent the ALPIDE chips in the 3D system, a plane model was selected with the same aspect ratio as the actual chips. Registered hits are marked on the planes by tracing the hit position via the x/y hit coordinates, and placing a marker. The planes have been textured to resemble the real ALPIDE sensor, as shown in Figure 4.

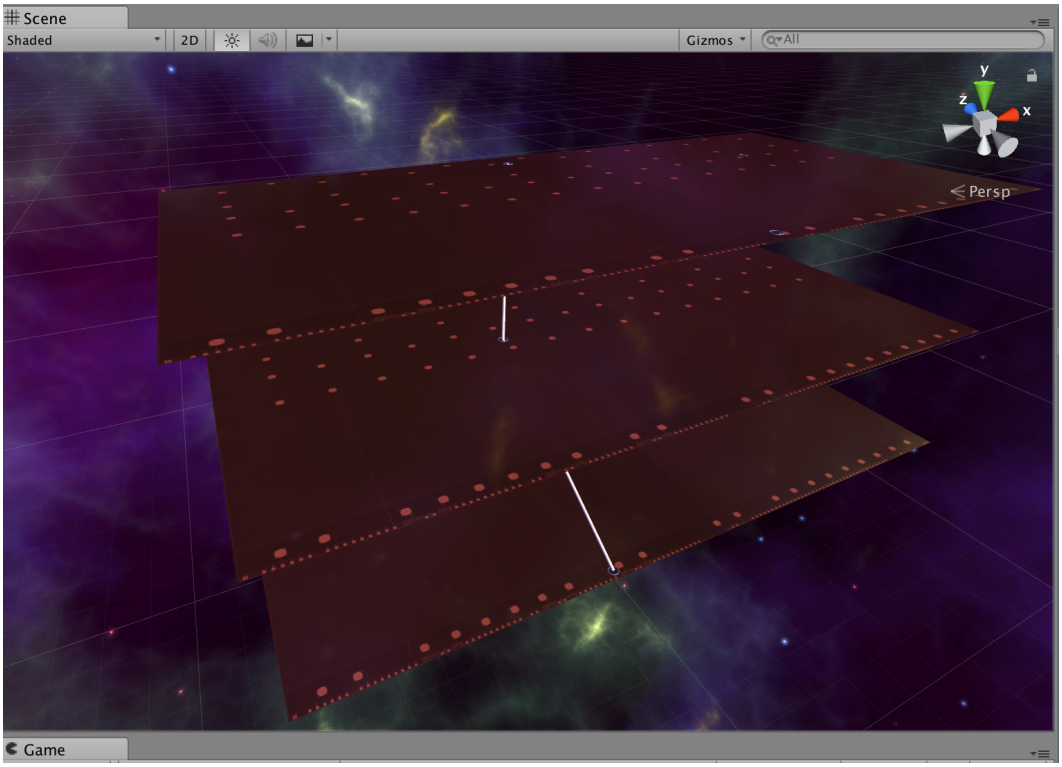

Figure 4: Virtual ALPIDE sensors

In order to make a complete representation of an Arduino/Alpide stack, simple models for the shield and Arduino were designed in Autodesk Maya, as shown in Figure 5.

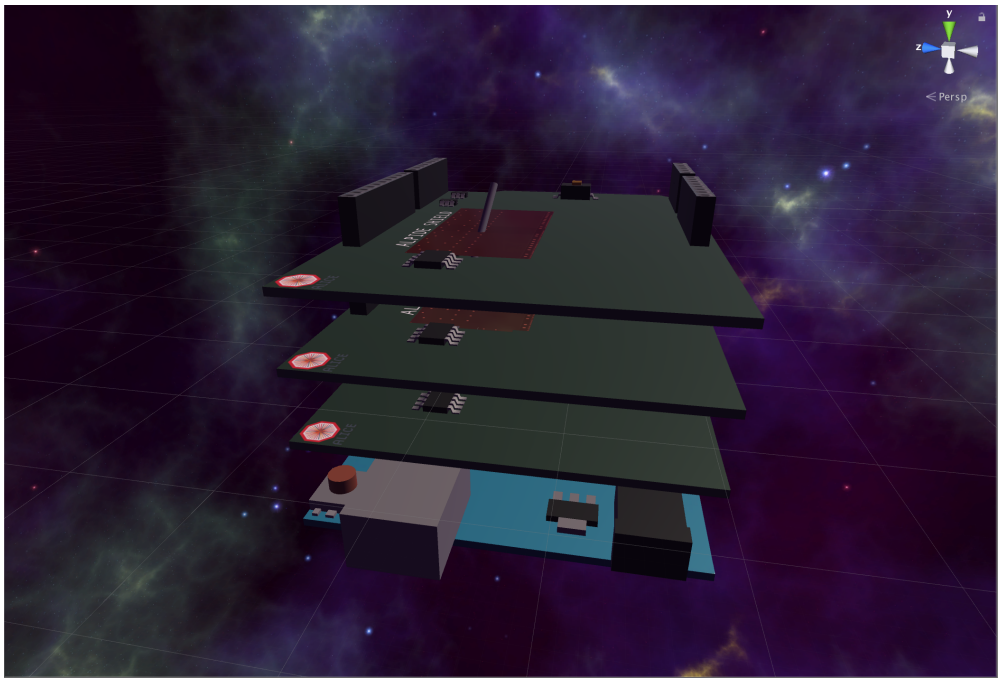

Figure 5: ALPIDE Shield and Arduino Models

3.2.4 Filtering and Masking Data

Before the data is marked on the planes, it first has to pass through a filtering process. Some of the pixels in each chip register frequent hits, which are not caused by actual muons, and hence can be described as being 'noisy'. These pixels need to be filtered out and ignored when attempting to deduce a muon track.

In order to mask these out, a simple procedure has been implemented. A set amount of data events are collected, and for each plane, the pixel hit frequencies are recorded. If a certain pixel is repeating registering hits (above a set threshold), it is marked and removed from all calculations.

3.2.5 Clustering

One of the main calculations taking place is the identification of clusters. A particle hit can cause multiple pixels to fire (depending on its direction of flight, momentum, species, etc) which are adjacent to one another. It is highly unlikely that two pixels adjacent to each other will fire due to noise, especially after filtering. A clustering algorithm is in place which groups pixels adjacent to one another into clusters.

The centre of gravity is calculated from each cluster, which acts as the hit position for the particle in question.

3.2.6 Detecting Tracks

The second part of the data analysis is the determination of a track's existence and trajectory. To do this, each cluster hit position is analyzed in relation to all the other cluster hits in the rest of the planes. A particle is expected to pass straight through each detector, since the detectors only cause very slight deviations in trajectory.

With this knowledge, one can say that a track has been detected when three cluster hits line up across 3 or more planes. It is highly unlikely that this happens due to noise, given the large amount of pixels in each plane.

To test for this condition, the vector dot product is utilised. For the case of three planes, the dot product can be used to deduce the angle between the two vectors formed by linking clusters hits in adjoining planes. This angle can be used to represent the deviation from a straight line.

For four planes, two dot products need to be calculated for the two vectors formed between an inner plane and its adjacent outer plane. The total deviation is taken as the sum of these two products.

Vectors which have a deviation which is less than a set threshold can be said to represent a particle track.

Once a particle track is detected, it is represented on screen via an actual track. This extends through each plane and persists for a user-selected amount of time, as in Figure 6.

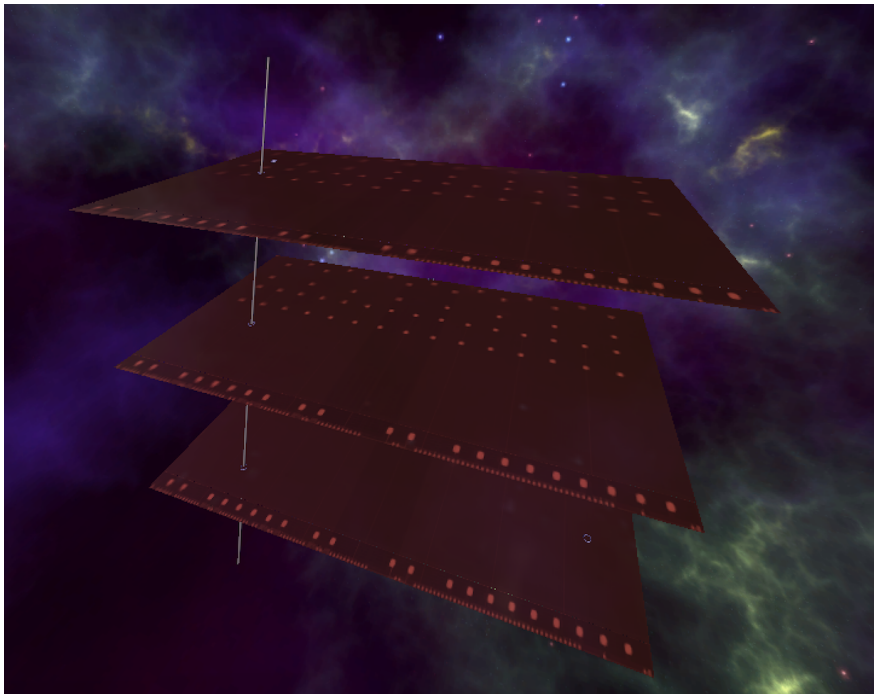

Figure 6: Track formation for a near-perfect straight line

3.2.7 Data Storage

Apart from the visualisation of data, the system also takes care of saving the original input data, the mask in use and the cluster data found. These data streams are seperated into different files, which can be activated/deactivated by the user. The data is saved automatically when received, thanks to a dedicated coroutine process.

3.3 User Interface

The User Interface (UI) has been designed to be as uncluttered as possible. The Main Menu (Figure 7) presents the user with two main options - Measure and Mask Noise.

'Measure' initialises the system to load a saved noise mask and start gathering and displaying data direct from a connected ALPIDE stack.

'Mask Noise' makes the system gather a set number of events, and select maskable pixels depending on their frequency.

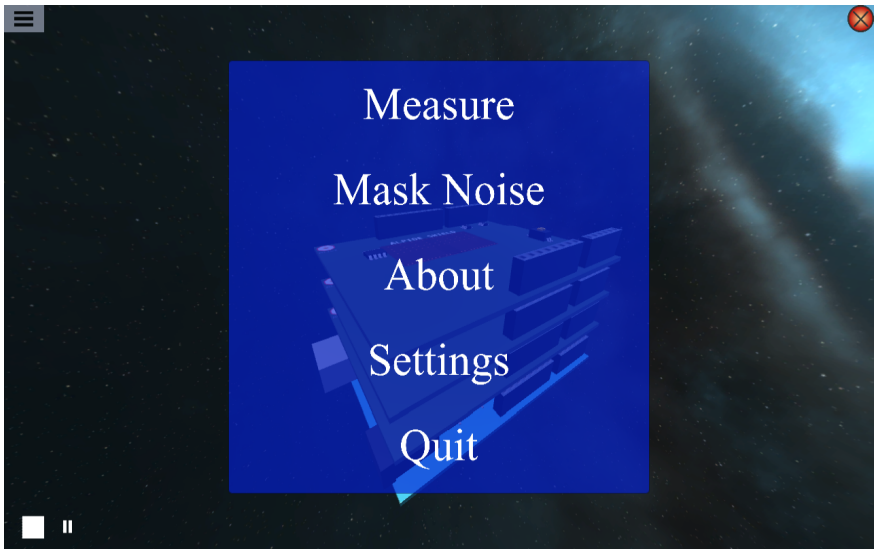

Figure 7: Main Menu

A settings page (Figure 8) allows the user to tweak the noise thresholds, set the height of each plane (these affect the track calculations), select the number of planes and vary the serial port timeout.

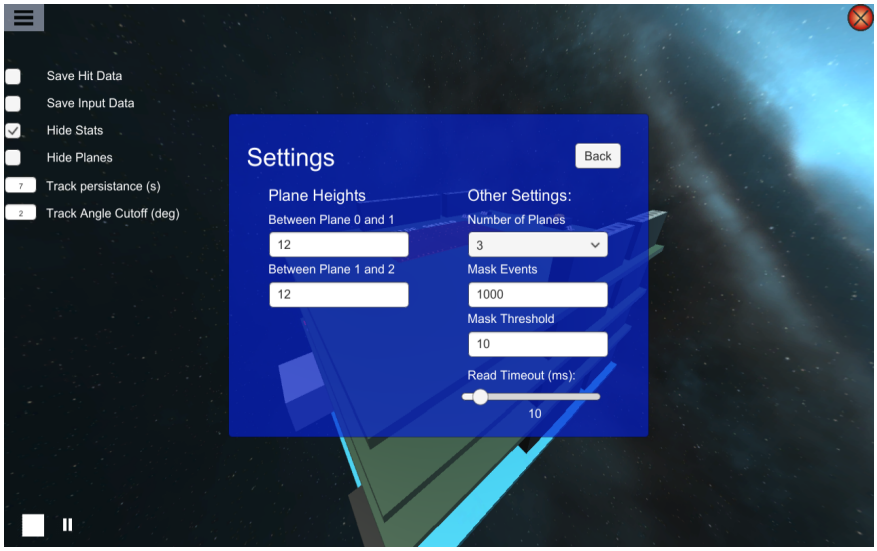

Figure 8: Settings Page

A number of settings are also available via drop down menu at the top left of the screen. The following settings can be changed during measurement via this menu:

- Data Saving - Selecting which data should be saved to file at which point in time.
- Hide/Show statistics - These statistics show the current hits, the number of tracks found and the current track angle deviation found.
- Hide/Show Shields - This allows one to view the ALPIDE chips more easily.
- Track persistance - Determines how long a track remains on screen.
- Track angle threshold - Determines the angle of deviation acceptable for a track to be formed.

A user can also pause/play and stop measurement at any time through the dedicated buttons on the bottom left of the screen.

4 Results

The full working system is shown in Figure 9.

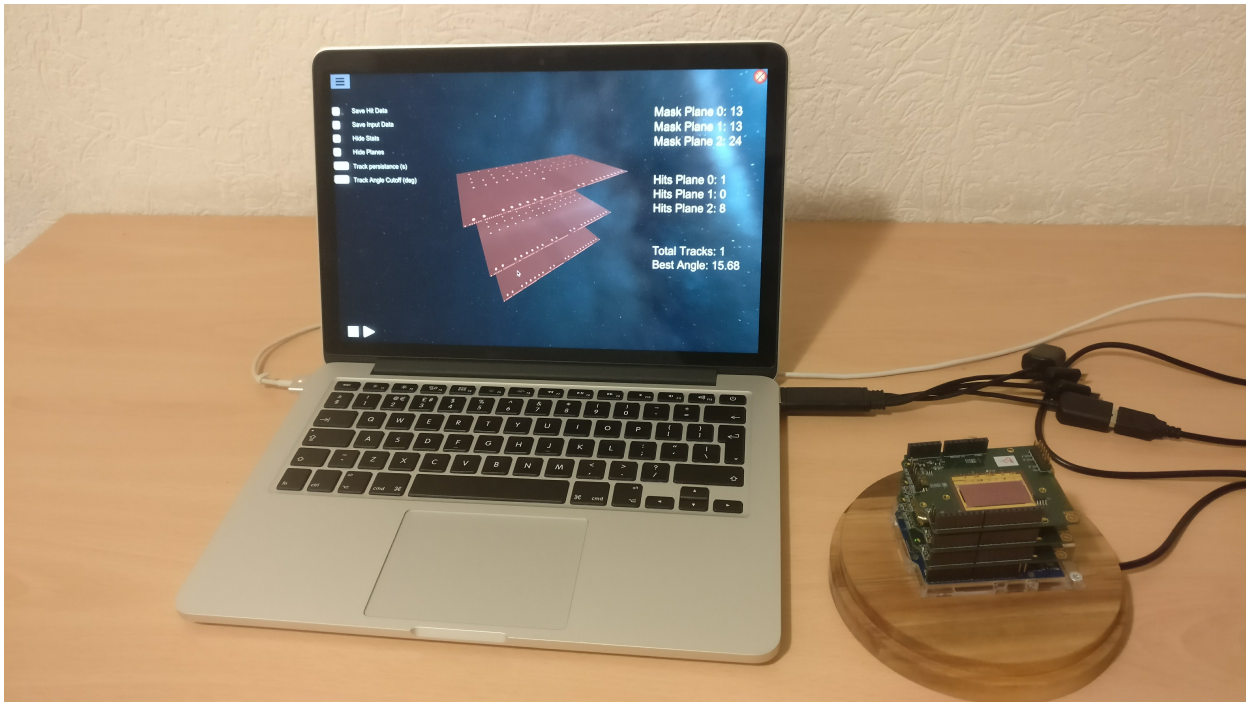

Figure 9: Full System

The system can currently support 3 or 4 planes on the stack, and can run unattended for long periods of time. The application is supported and has been tested on Mac, Windows and Linux as an executable file.

The rule of thumb for muon rates is 1 per cm^2/minute. The ALPIDE sensor has a surface area of 4 cm^2 and so one would expect it to receive 4 muons/ minute. This is not the case, however, since the way a track is detected via the stack of sensors limits a detectable

muon to a certain range of trajectories. The actual rate observed is closer to 1-2 muons every 2 minute; assuming a maximum angle of deviation of 2 degrees.

5 Issues and Improvements

5.1 Hardware

While the system has been coded to support 4 planes, official commissioning has not yet been carried out, due to some hardware issues with one of the planes in the lab. Further testing is required with another plane to fully confirm the functionality of the system with 4 planes.

5.2 Compatibility Issues

Whilst the application has been tested and works perfectly on Mac and Windows, the system is not fully compatible with every version of Linux. The application is fully functional on Ubuntu 10.04 and above, but encounters issues with other Linux distributions such as CentOS. As yet no fix has been found to address these compatibility issues.

5.3 Data Volume

The system is currently geared to accept small volumes of hits at a time, which is what is expected from atmospheric radiation. If, however, the system were to be used with a radiation source causing a lot of hits, the system will fail since it is not built to support a high volume(hundreds or more) of hits. The issue is in the manner the data is being stored before being sent, and not due to serial port issues.

In order to rectify this, both the Arduino code and Unity code need to be overhauled to use dynamic memory systems to accept larger amounts of data at a time.

6 Conclusion

The goal of this project was reached; a standalone graphical application was built which successfully interfaces with an ALPIDE sensor stack. It visually recognises and represents tracks passing through the stack whilst also showing the current hit positions sent from the Arduino stack. This functionality is adjustable through a custom UI and the ability to save data for future analysis.

This setup allows for further expansion and case studies to be set up as a learning tool for students eager to enter the world of particle physics. Moreover, it is a project which allows cutting-edge MAPS technology to be distributed and utilised at a very cheap price - improving the awareness and accessibility of this technology.

References

[1] Technical Design Report for the Upgrade of the ALICE Inner Tracking System at: http://iopscience.iop.org/article/10.1088/0954-3899/41/8/087002/meta

[2] ALPIDE Chip Overview: http://inspirehep.net/record/1513530

[3] ALPIDE Operations Manual V0.3 (2016) - Internal Communication

[4] Cosmic Radiation at Sea Level at: https://arxiv.org/abs/1208.1171

[5] Unity Official Website: https://unity3d.com

[6] JSON Protocol Description: http://www.json.org