Enhanced adjacency trigger algorithm

Combined PP/DS WG meeting

May 7th, 2024

Hamza Amar Es-sghir

Animesh Chatterjee

DEEP UNDERGROUND NEUTRINO EXPERIMENT

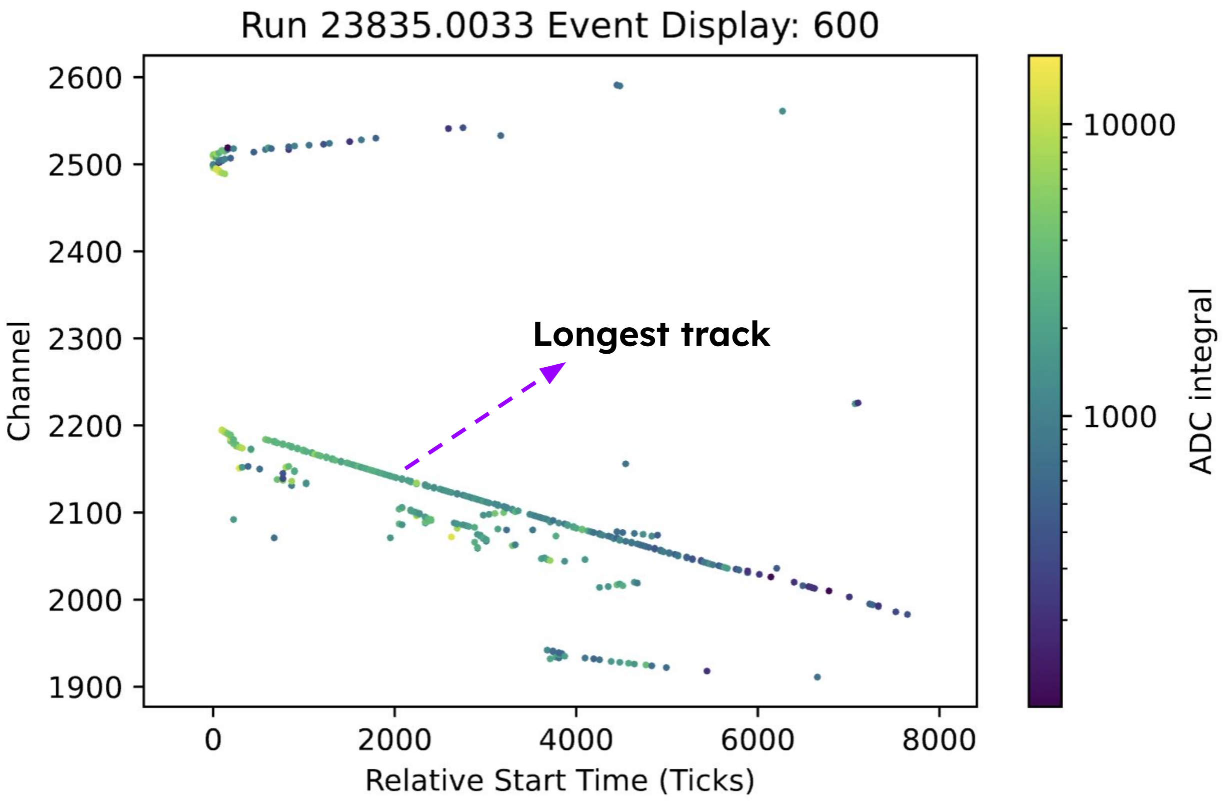
IFIC INSTITUT DE FÍSICA CORPUSCULAR

CSIC CONSEJO SUPERIOR DE INVESTIGACIONES CIENTÍFICAS

VNIVERSITAT DE VALÈNCIA

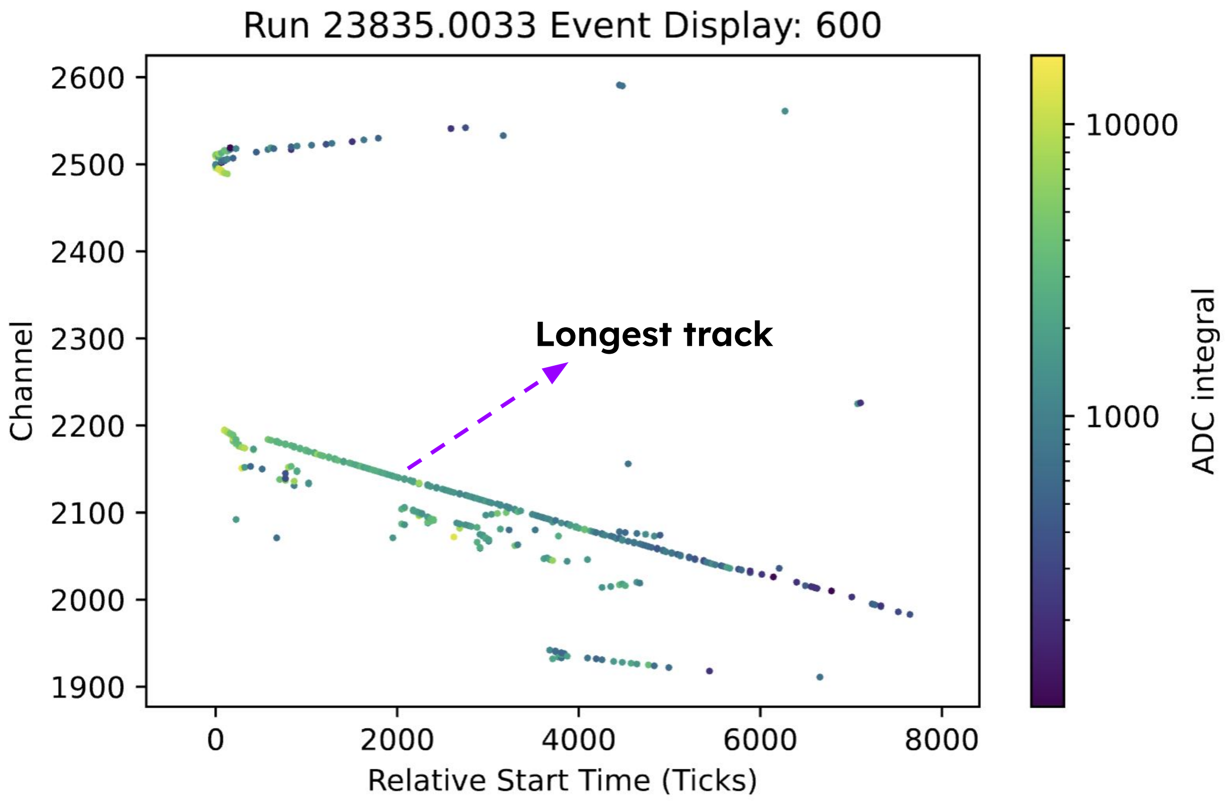
Motivation

- So far, current adjacency algorithm used in the *Horizontal Muon algorithm* saves all TPs inside the time window provided the adjacency condition is met.
- **Goal**: remove background within the time window requiring a threshold.
- The use of other TP information was explored.
- With **ADC integral** or **ADC peak**, background cannot be removed keeping track TPs.



Motivation

- One way to clean up the TA and select the longest track would be using not only channel, but also time information.
- Everytime channel adjacency is checked, there could be an extra condition. If TP time difference is less than a given tolerance we add TP information to a vector of TPs.
- Generate TAs with filtered TPs.
- Time tolerance must be determined checking TP information of tracks of interest.



Flowchart diagram for adjacency trigger algorithm

Trigger Primitives (TPs)

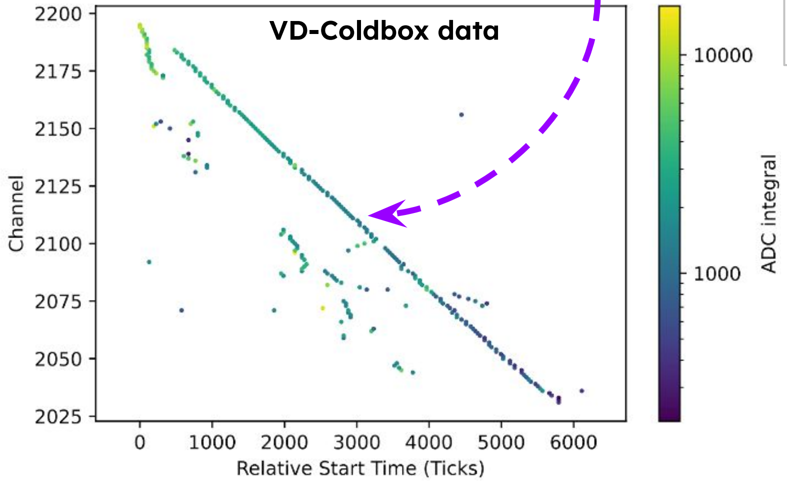
Adjacency trigger (based on adjacent hit collection wires threshold)

Goal: find longest track within a time window of 8000 ticks. Time and channel information are combined

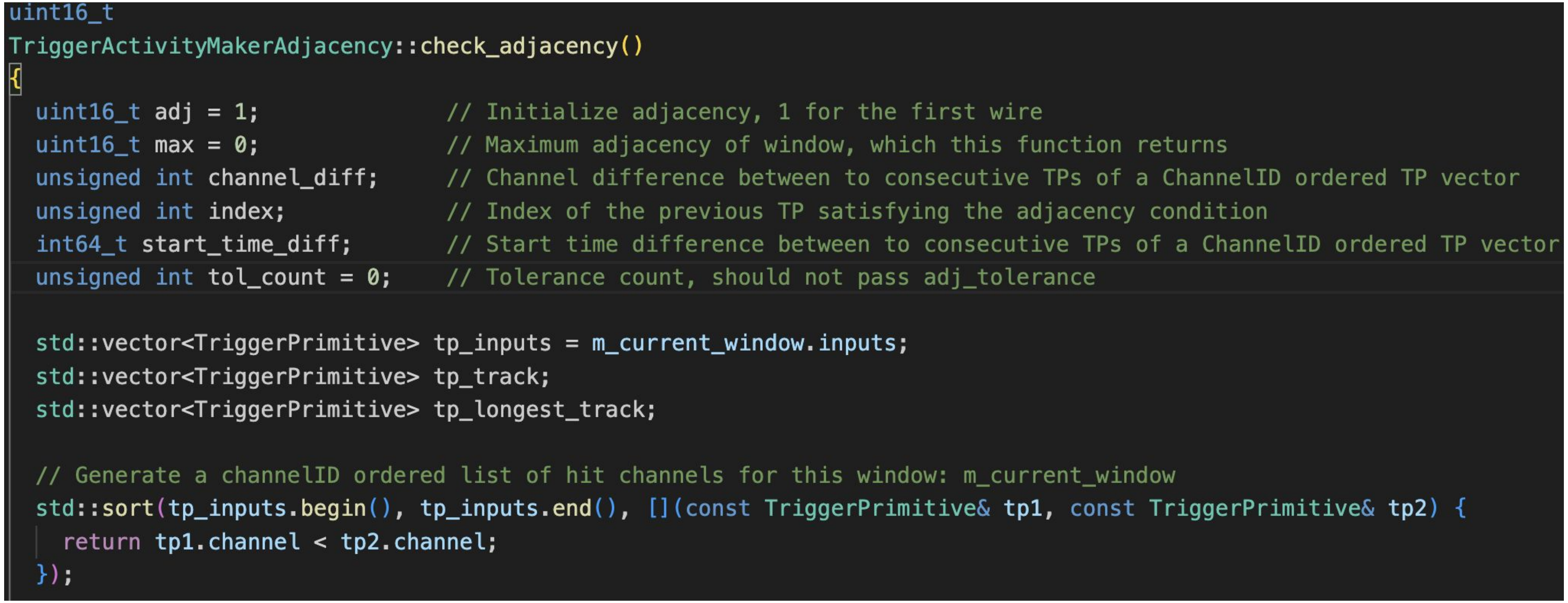
Channel adjacency TPs at adjacent channels. Channel tolerance accounts for TP skips.

Time tolerance TPs must also meet a time tolerance to be considered adjacent, i.e. of the same track.

Trigger Activity (TA)

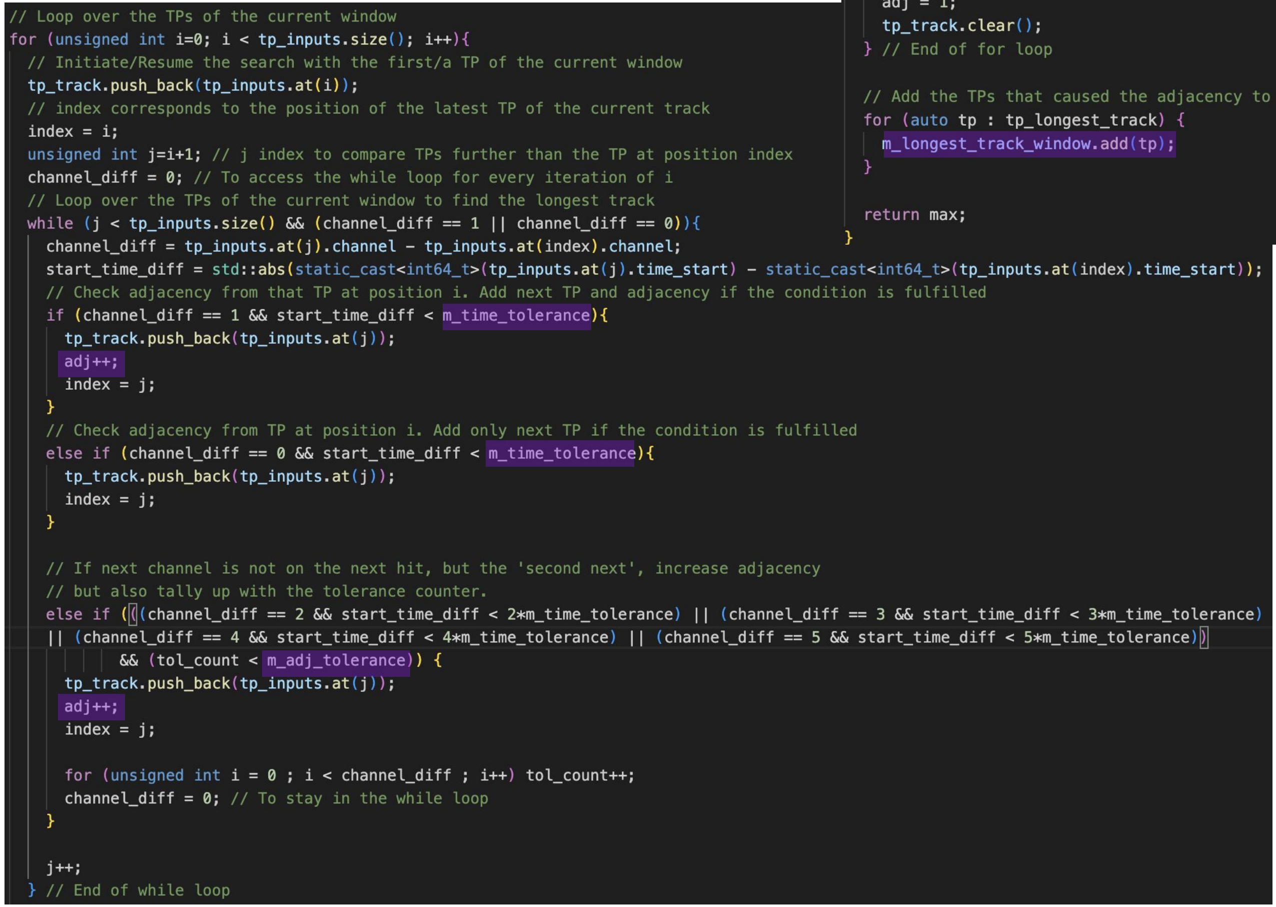


New algorithm logic

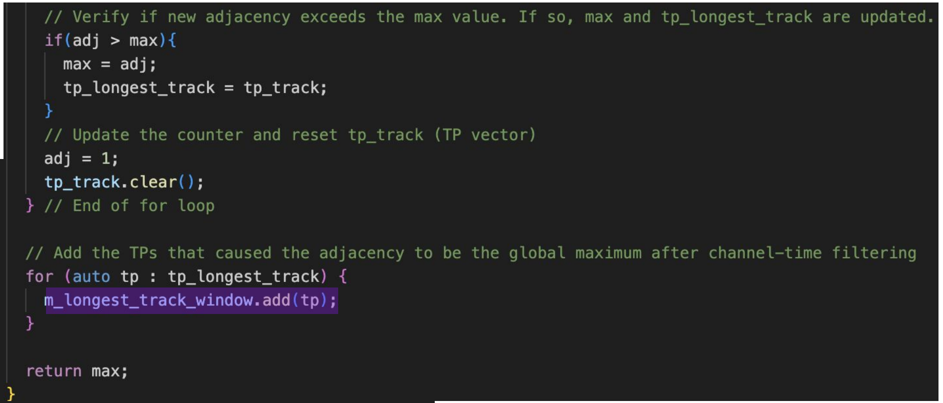


```cpp
uint16_t
TriggerActivityMakerAdjacency::check_adjacency()
{
  uint16_t adj = 1;              // Initialize adjacency, 1 for the first wire
  uint16_t max = 0;              // Maximum adjacency of window, which this function returns
  unsigned int channel_diff;     // Channel difference between to consecutive TPs of a ChannelID ordered TP vector
  unsigned int index;            // Index of the previous TP satisfying the adjacency condition
  int64_t start_time_diff;       // Start time difference between to consecutive TPs of a ChannelID ordered TP vector
  unsigned int tol_count = 0;    // Tolerance count, should not pass adj_tolerance

  std::vector<TriggerPrimitive> tp_inputs = m_current_window.inputs;
  std::vector<TriggerPrimitive> tp_track;
  std::vector<TriggerPrimitive> tp_longest_track;

  // Generate a channelID ordered list of hit channels for this window: m_current_window
  std::sort(tp_inputs.begin(), tp_inputs.end(), [](const TriggerPrimitive& tp1, const TriggerPrimitive& tp2) {
    return tp1.channel < tp2.channel;
  });
```

New algorithm logic



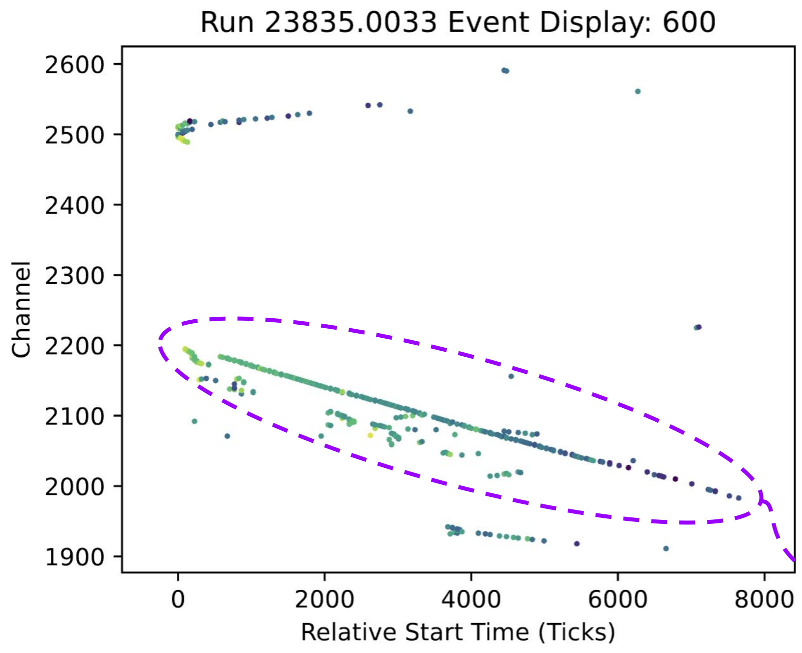
```
// Loop over the TPs of the current window
for (unsigned int i=0; i < tp_inputs.size(); i++){
  // Initiate/Resume the search with the first/a TP of the current window
  tp_track.push_back(tp_inputs.at(i));
  // index corresponds to the position of the latest TP of the current track
  index = i;
  unsigned int j=i+1; // j index to compare TPs further than the TP at position index
  channel_diff = 0; // To access the while loop for every iteration of i
  // Loop over the TPs of the current window to find the longest track
  while (j < tp_inputs.size() && (channel_diff == 1 || channel_diff == 0)){
    channel_diff = tp_inputs.at(j).channel - tp_inputs.at(index).channel;
    start_time_diff = std::abs(static_cast<int64_t>(tp_inputs.at(j).time_start) - static_cast<int64_t>(tp_inputs.at(index).time_start));
    // Check adjacency from that TP at position i. Add next TP and adjacency if the condition is fulfilled
    if (channel_diff == 1 && start_time_diff < m_time_tolerance){
      tp_track.push_back(tp_inputs.at(j));
      adj++;
      index = j;
    }
    // Check adjacency from TP at position i. Add only next TP if the condition is fulfilled
    else if (channel_diff == 0 && start_time_diff < m_time_tolerance){
      tp_track.push_back(tp_inputs.at(j));
      index = j;
    }

    // If next channel is not on the next hit, but the 'second next', increase adjacency
    // but also tally up with the tolerance counter.
    else if (((channel_diff == 2 && start_time_diff < 2*m_time_tolerance) || (channel_diff == 3 && start_time_diff < 3*m_time_tolerance)
    || (channel_diff == 4 && start_time_diff < 4*m_time_tolerance) || (channel_diff == 5 && start_time_diff < 5*m_time_tolerance))
          && (tol_count < m_adj_tolerance)) {
      tp_track.push_back(tp_inputs.at(j));
      adj++;
      index = j;

      for (unsigned int i = 0 ; i < channel_diff ; i++) tol_count++;
      channel_diff = 0; // To stay in the while loop
    }

    j++;
  } // End of while loop
```



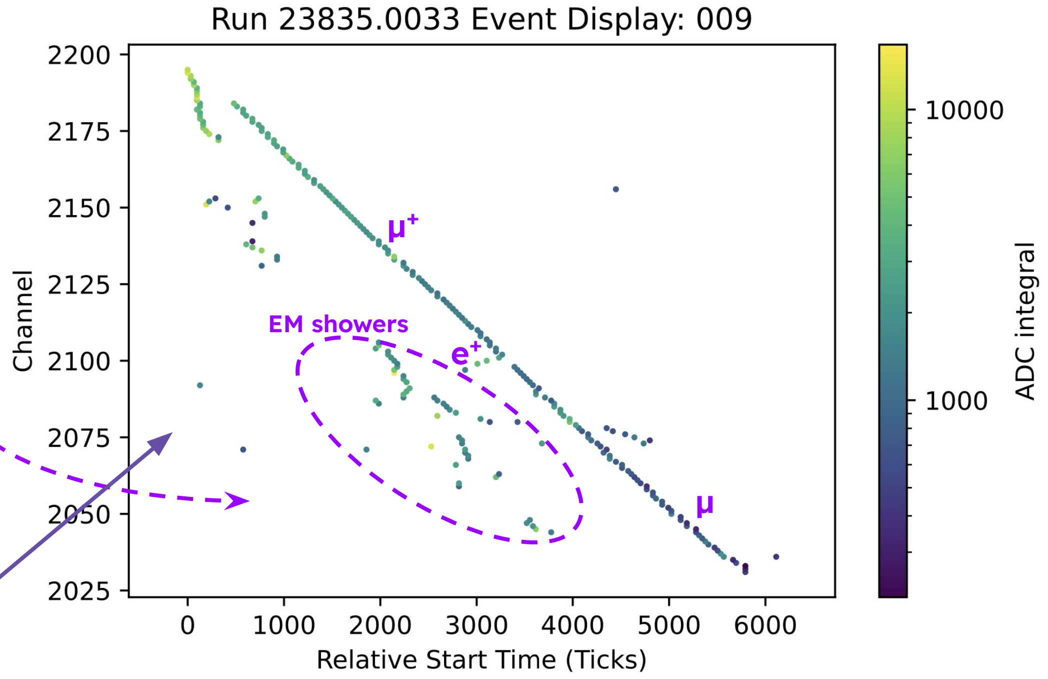
```
    // Verify if new adjacency exceeds the max value. If so, max and tp_longest_track are updated.
    if(adj > max){
      max = adj;
      tp_longest_track = tp_track;
    }
    // Update the counter and reset tp_track (TP vector)
    adj = 1;
    tp_track.clear();
  } // End of for loop

  // Add the TPs that caused the adjacency to be the global maximum after channel-time filtering
  for (auto tp : tp_longest_track) {
    m_longest_track_window.add(tp);
  }

  return max;
}
```

Variable	Role	Name
Channel adjacency	Accounts for channel multiplicity	*adj*
Time tolerance	Max time difference admitted for adjacent channels	*m_time_tolerance*
Adjacency tolerance	Permitted channel skips	*m_adj_tolerance*

Cosmic ray rejection: new trigger algorithm performance

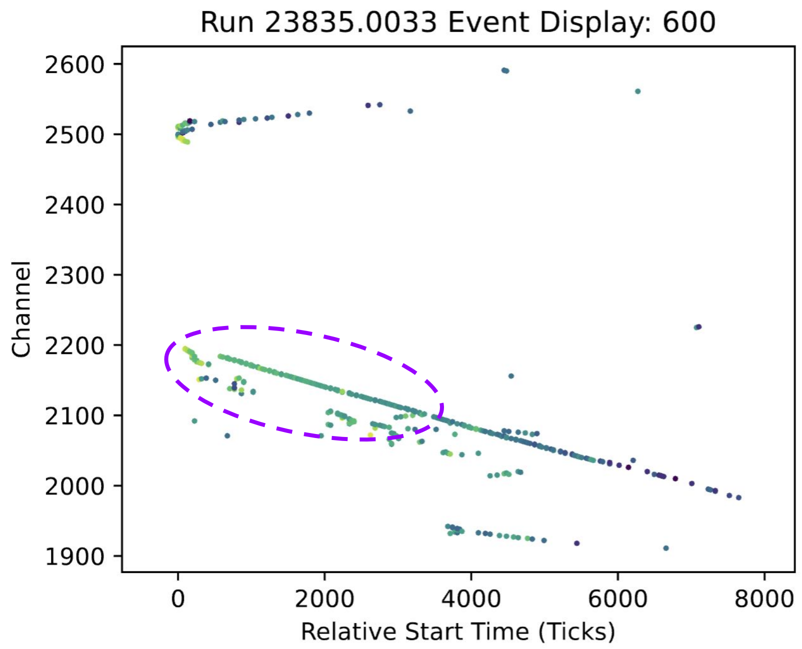


Before changes

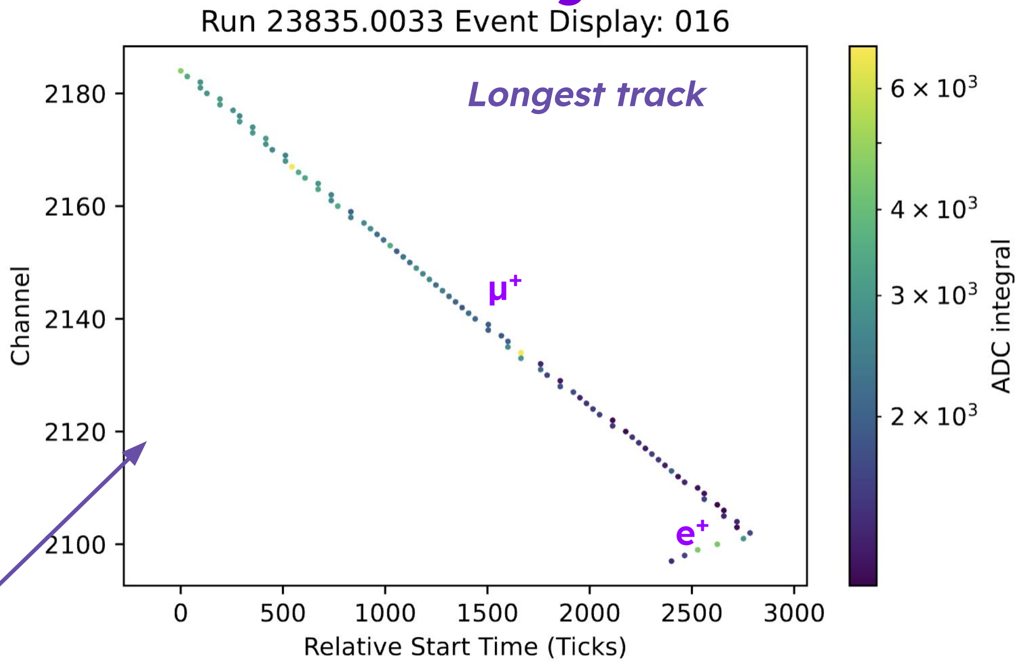


Inputs	**Adjacency threshold**: 100 channels **Time tolerance**: time window length **Adjacency tolerance**: 0 channels

Cosmic ray rejection: new trigger algorithm performance

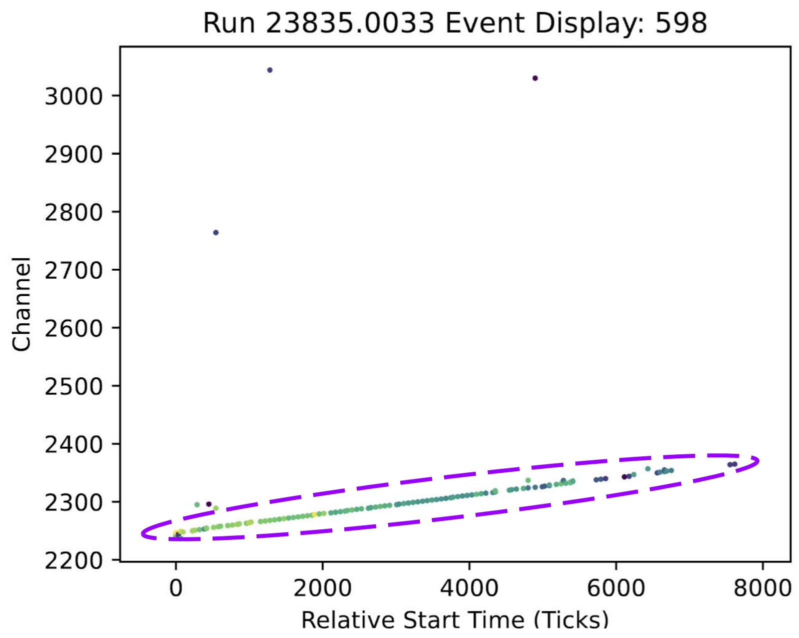


After changes



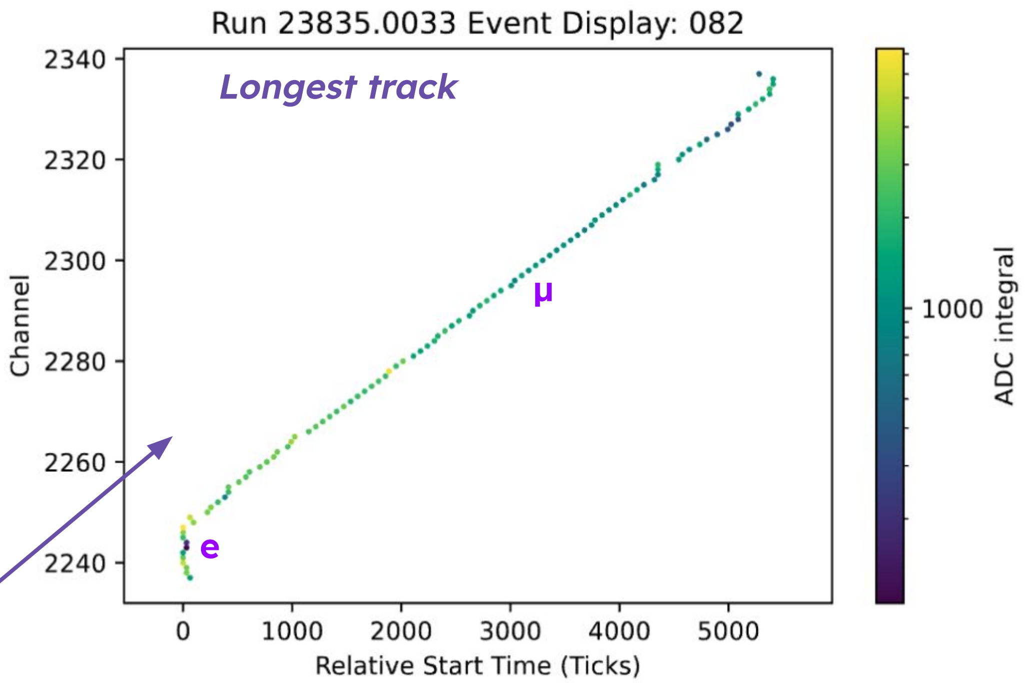
Inputs	
	Adjacency threshold: 80 channels
	Time tolerance: 120 ticks
	Adjacency tolerance: 5 channels

Cosmic ray rejection: new trigger algorithm performance



Inputs	**Adjacency threshold**: 100 channels **Time tolerance**: 100 ticks **Adjacency tolerance**: 3 channels

After changes



Conclusions

- Combination of both TP channel and time information has proven an efficient option to clean up tracks of interest.

- Instead of saving the full time window, solely the longest track is saved.

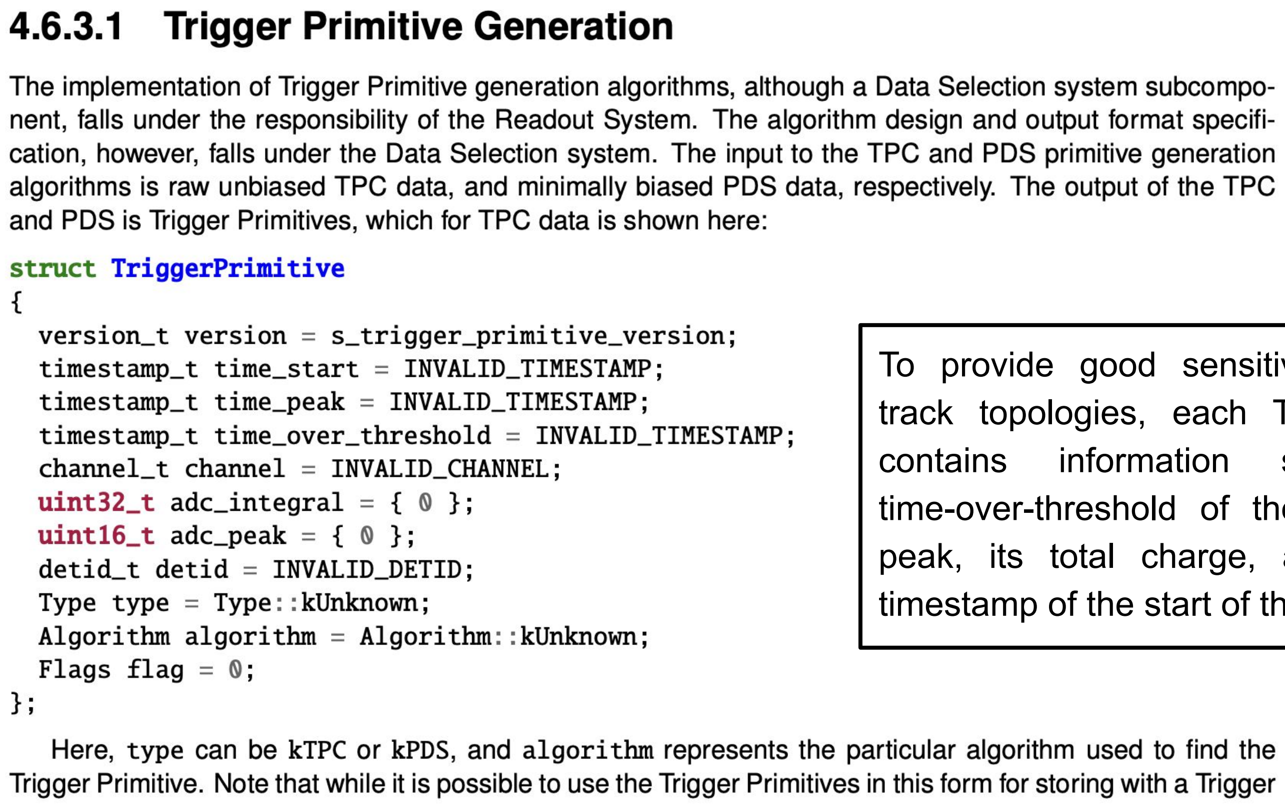
- Relevant parameters to be considered in the config file are:
  - Adjacency threshold.
  - Time tolerance.
  - Adjacency tolerance.

- Same results as filtering by channel only are reproduced setting a time tolerance larger than the time window length.

- In case we wanted to save more tracks, code may be adjusted to fulfill the requirements.

Backup

Trigger Primitive structure



4.6.3.1 Trigger Primitive Generation

The implementation of Trigger Primitive generation algorithms, although a Data Selection system subcomponent, falls under the responsibility of the Readout System. The algorithm design and output format specification, however, falls under the Data Selection system. The input to the TPC and PDS primitive generation algorithms is raw unbiased TPC data, and minimally biased PDS data, respectively. The output of the TPC and PDS is Trigger Primitives, which for TPC data is shown here:

```
struct TriggerPrimitive
{
  version_t version = s_trigger_primitive_version;
  timestamp_t time_start = INVALID_TIMESTAMP;
  timestamp_t time_peak = INVALID_TIMESTAMP;
  timestamp_t time_over_threshold = INVALID_TIMESTAMP;
  channel_t channel = INVALID_CHANNEL;
  uint32_t adc_integral = { 0 };
  uint16_t adc_peak = { 0 };
  detid_t detid = INVALID_DETID;
  Type type = Type::kUnknown;
  Algorithm algorithm = Algorithm::kUnknown;
  Flags flag = 0;
};
```

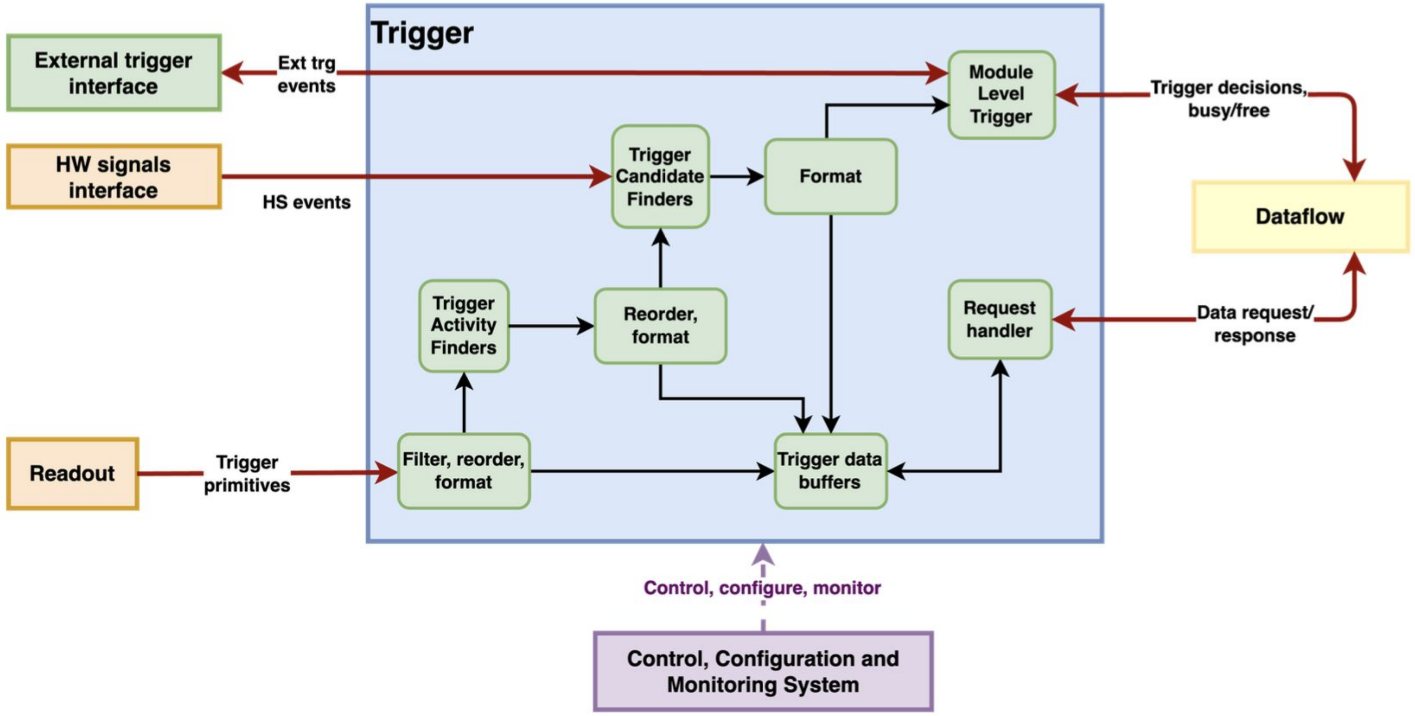
Here, `type` can be `kTPC` or `kPDS`, and `algorithm` represents the particular algorithm used to find the Trigger Primitive. Note that while it is possible to use the Trigger Primitives in this form for storing with a Trigger

To provide good sensitivity to different track topologies, each Trigger Primitive contains information such as the time-over-threshold of the waveform, its peak, its total charge, as well as the timestamp of the start of the waveform.

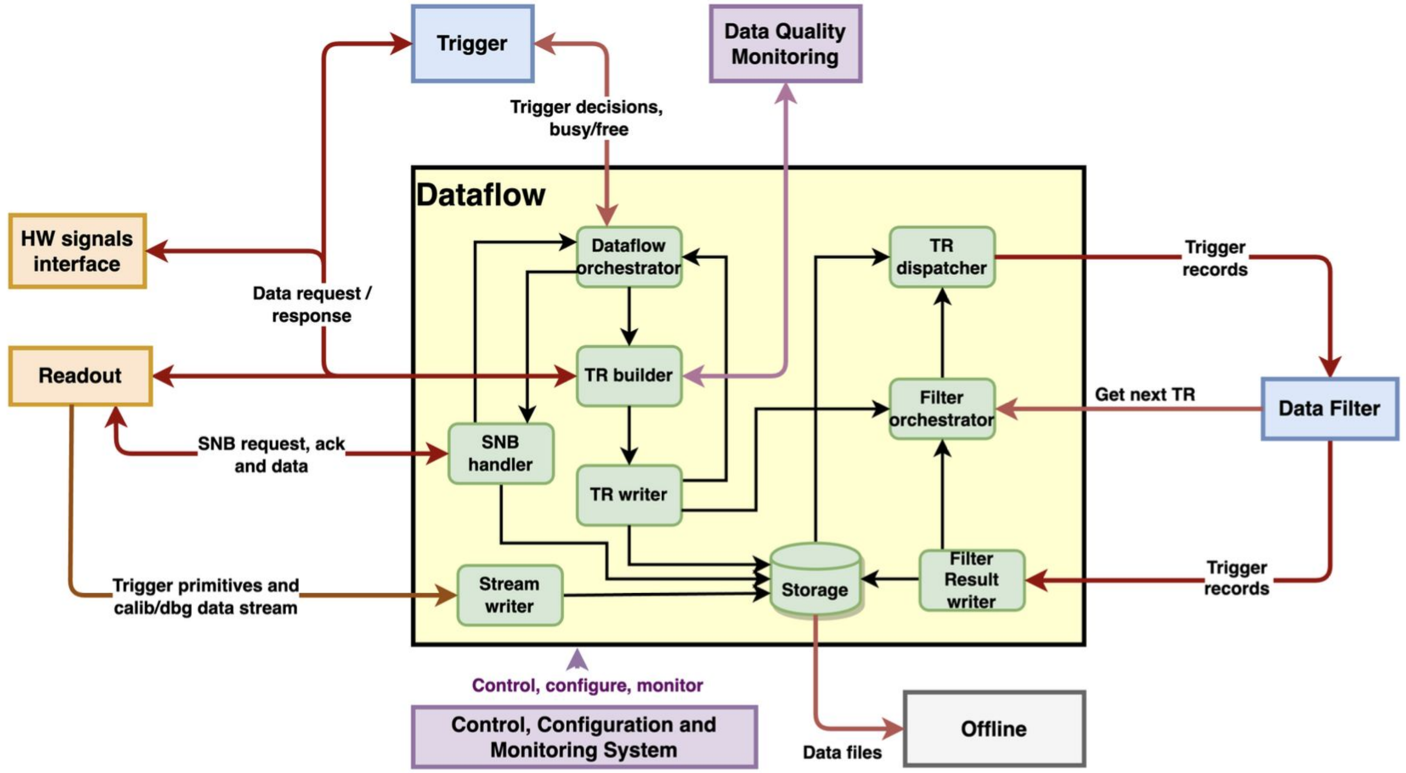
Trigger Activity structure

```
struct TriggerActivity
{
  static constexpr version_t s_trigger_activity_version = 1;

  version_t version = s_trigger_activity_version;
  timestamp_t time_start = INVALID_TIMESTAMP;
  timestamp_t time_end = INVALID_TIMESTAMP;
  timestamp_t time_peak = INVALID_TIMESTAMP;
  timestamp_t time_activity = INVALID_TIMESTAMP;
  channel_t channel_start = INVALID_CHANNEL;
  channel_t channel_end = INVALID_CHANNEL;
  channel_t channel_peak = INVALID_CHANNEL;
  uint64_t adc_integral = 0;
  uint16_t adc_peak = 0;
  detid_t detid = INVALID_DETID;
  Type type = Type::kUnknown;
  Algorithm algorithm = Algorithm::kUnknown;
};
```

Trigger components and their interaction with other DAQ sub-systems



Interfaces of Dataflow components with other TDAQ sub-systems



Readout planes for Horizontal drift module

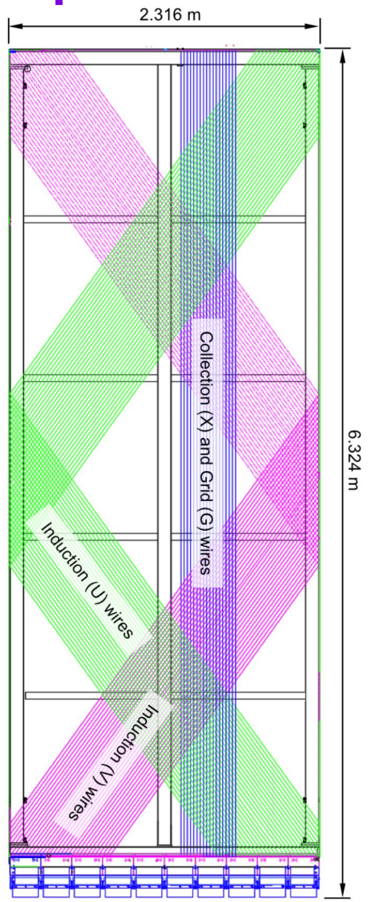
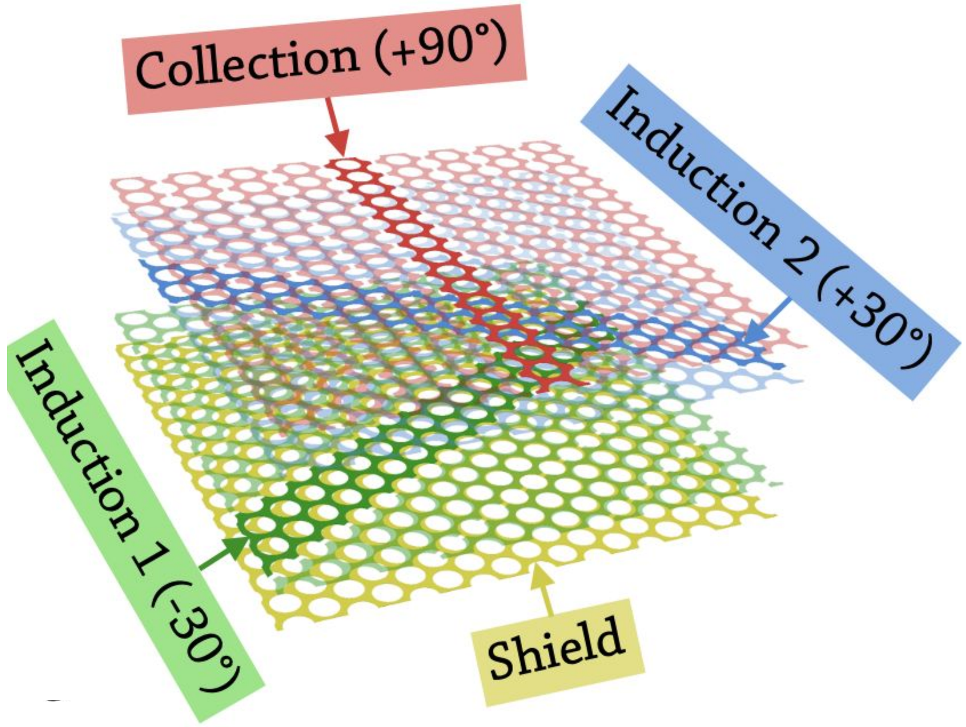


Table 2.3: APA design parameters

Parameter	Value
Active height	5.984 m
Active width	2.300 m
Wire pitch (U, V)	4.7 mm
Wire pitch (X, G)	4.8 mm
Wire pitch tolerance	$\pm$0.5 mm
Wire plane spacing	4.8 mm
Wire plane spacing tolerance	$\pm$0.5 mm
Wire Angle (w.r.t. vertical) (U, V)	$\pm$35.7 °
Wire Angle (w.r.t. vertical) (X, G)	0 °
Number of wires / APA	960 (X), 960 (G), 800 (U), 800 (V)
Number of electronic channels / APA	2560
Wire material	beryllium copper (CuBe)
Wire diameter	152 μm

Readout planes for Vertical drift module


Collection (+90°)
Induction 2 (+30°)
Induction 1 (-30°)
Shield