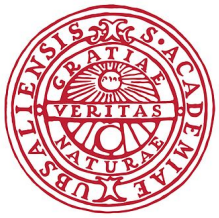


UPPSALA
UNIVERSITET

Coordinate conversion for the Hough transform

Edvin Eriksson
Supervisor: Richard Brenner, HEP, Uppsala University
Subject Reader: Rebeca Gonzalez Suarez, HEP, Uppsala University
Exam Coordinator: Matthias Weiszflog

June 2021

Abstract

This thesis attempts to develop a conversion algorithm between local coordinates in constituent detector modules and global coordinates encompassing the whole detector structure in a generic detector. The thesis is a part of preparatory work for studying the Hough Transform as a means of track reconstruction in the level-1 hardware trigger in the upgraded trigger and data acquisition (TDAQ) system in the phase 2 upgrade of the ATLAS detector at CERN. The upgrades being made are to withstand much more extreme conditions that come with the high-Luminosity Large Hadron Collider (HL-LHC).

Two algorithms have been made and then implemented in Python scripts to test their feasibility and to compare them against each-other. The Rotation algorithm uses several rotations to correctly place the local coordinates in the global system. The second, the Shear algorithm, simplifies the process into two shears and one rotation, using the small angle approximation. Both algorithms need to be extended to work with more parts of the detector to be considered complete. Despite having lower maximum precision the second algorithm is considered the most promising attempt, since it is much less sensitive to the truncation error that results from working in an integer environment, which is a requirement for use in FPGAs.

Sammanfattning

I denna uppsats görs ett försök att skapa en omvandlingsalgoritm mellan lokala koordinater i konstituerande detektormoduler och globala koordinater i hela detektorstrukturen för en generisk detektor. Uppsatsen är en del i förberedande arbete för att undersöka hur Houghtransformen kan användas för spårrekonstruktion i den hårdvarubaserade *level-1 triggern* i det uppgraderade trigger- och datainsamlingssystemet (TDAQ) i fas två-uppgraderingen av ATLAS detektorn vid CERN. Uppgraderingarna som görs är för att kunna utstå de mycket mer extrema förhållanden som medförs av högluminositetsuppgraderingen av *Large Hadron Collider* (HL-LHC).

Två algoritmer har skapats och implementerats i Pythonskript för att testa genomförbarhet och för att jämföra med varandra. Rotationsalgoritmen använder ett antal rotationer för att korrekt placera ut de lokala koordinaterna i det globala systemet. Den andra, Skjuvalgortimen, förenklar processen till två skjuvningar och en rotation med hjälp av liten vinkel-approximationen. Båda algoritmerna behöver utökas för att fungera för fler delar av detektorn för att anses kompletta. Trots lägre maximal precision bedöms den andra algoritmen vara det mest lovande försöket, eftersom den är mycket mindre känslig för trunkeringsfelet som kommer av att arbeta i en heltalsmiljö, som är ett krav för FPGA-implementationen.

Contents

1 Introduction

The current hardware of the Large Hadron Collider (LHC) at CERN is near its limit both in terms of capacity and radiation damage [1]. The most important product of the LHC is high-energy particle collisions, for measurements at the various experiments connected to it. Therefore one of the most important measures of the LHC's performance is the rate of collisions, both instantaneously and over time, usually measured in collision rate per unit area, *luminosity*, and the (over time) *integrated luminosity* [1]. While the LHC is currently shut down (Long Shut-Down 2, LS2) for maintenance and some smaller upgrades, when it starts again for Run 3 in 2022 it is planned to run at the current hardware's maximum luminosity until 2024 when it will reach an accumulated integrated luminosity of 350 fb^{-1} (for reference Run 1 data totalled 30 fb^{-1} and run 2 190 fb^{-1}) [1] and parts of the collider and detectors will have accrued so much radiation damage as to negatively effect performance [3].

Instead of just repairs Long Shut-Down 3 (LS3) from 2025 to 2027 will contain several technology updates and engineering upgrades that have been in the works since 2010 [1]. These will allow the LHC to reach a luminosity five times higher than currently feasible and over its decade of operation up to 4000 fb^{-1} integrated luminosity [1]. The upgraded machine is called the High Luminosity LHC (HL-LHC).

One of the detectors at the LHC is the ATLAS detector, as mentioned above parts of it are also starting to receive too much radiation damage. It is also not engineered to handle the increased collision rate of the HL-LHC, which is why it too will undergo several upgrades [2][3].

Part of the problem, which will be further exacerbated by the HL-LHC, with recording data from HL-LHC events is the amount of data; the rate of beam crossings is 40 MHz, each of which generates a *pileup* of on average 200 proton-proton interactions, and each proton-proton collision generates several particles in turn [2]. There is simply too much data to process and all be saved to storage. Luckily a lot of collision events contain physics processes that are already well understood and those that require further exploration have signatures that can be used as conditions (or *triggers*) in an Event Filter to activate further processing and finally storage of the data in the so-called Trigger and Data Acquisition system (TDAQ) only for interesting events. [3] Filtering as early as possible in the detector is beneficial, since it reduces the load on all components that come after.

The innermost part of the ATLAS detector is the Inner Tracker (ITk), which records the position of charged particles moving through it at discrete points [2]. As part of the Event Filter the paths of charged particles will be

reconstructed using ITk data, by a system called Hardware Tracking for the Trigger (HTT) [3]. The HTT will be implemented using a combination of custom Associative Memory Application-Specific Integrated Circuits (AM-ASICs) and commercially available Field-Programmable Gate Arrays (FPGAs) [3]. An alternative method, which has been investigated by members of the Uppsala University ATLAS group [12][13], is using the *Hough Transform* [4] for track reconstruction. A system based on the Hough Transform could be implemented using only FPGAs, which would potentially reduce costs [12], however the Hough Transform requires associating the locations of datapoints from different parts of the ITk.

This thesis is an attempt to create an algorithm that associates the *local coordinates* of the datapoints with locations in the *global* view of the ITk for use with the Hough Transform in an FPGA implementation.

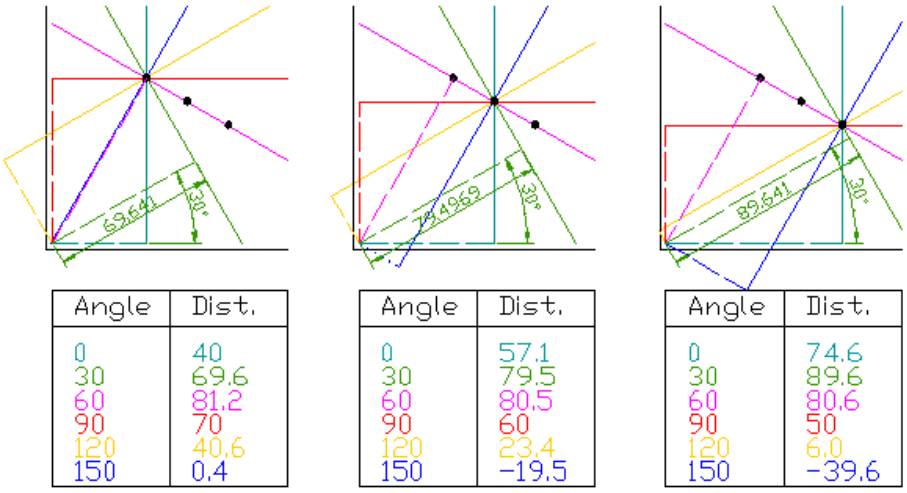
2 Background

2.1 The Hough Transform

The Hough Transform is a highly efficient tool for finding parametrized features in datasets, often lines or curves, first described by Hough in 1962 and then generalised by Hart and Duda in 1972 [4]. The equation at the centre of the Hough transform is

$$\rho = x\cos\theta + y\sin\theta \tag{1}$$

where θ is the angle of the normal of a line in the xy-plane and ρ is its algebraic distance from the origin. The $\rho\theta$-parametrization is called the normal parametrization by Hart and Duda [4]. The big advantage, relevant to particle paths, of the Hough transform is that points on a line in the xy-plane appear as lines intersecting a single point in the parameter space. In a discrete environment this allows the parameter space to act as a heat map for potential lines in the xy-plane, which in turn ends up being an efficient way of detecting co-linearity [4]. An illustrative example is made with the pink lines at $(\rho, \theta) = (60, 90)$ in Figure 1



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6

Figure 1: Demonstrating the principle of points along a line accumulating at a point. In the $\rho\theta$-plane the pink lines through the points all accumulate at the same point, telling us the lines are very similar, and therefore make a candidate for a path through all these points. Image: Public Domain, `commons.wikimedia.org/wiki/File:Hough_transform_diagram.png`

2.2 Inner Tracker structure

The proposed ITk for ATLAS consists of a barrel section with two mirrored endcap sections at each end. The detecting material in the ITk is placed in modules, spread out in the detector structure. There are two main types: pixel modules which contain a grid of silicon pixels, and strip modules which contain stacked strips of silicon. The modules are placed in layers, with the more granular pixel modules closer to the collision point of the two proton beams. There are a total of nine module layers in the barrel section, four outer strip layers, and five inner pixel layers. Near the central axis the endcaps consist of a complicated net of pixel modules, more adequately described in ch. 3 of the Technical Design Report [2]. Further out radially the strip layer endcaps have modules placed in petal shapes. Each module only records hits in its own local coordinate system. To restore some of the accuracy of the strip modules in the long direction of the strips strip modules are placed in paired layers with a small angle between the layers such that the strips of the different modules cross each-other.

For this thesis a similar, generic detector from a previous study by by Gradin, Mårtensson and Brenner [12] was used instead of the actual ATLAS layout. The generic detector has 5 pixel layers and 5 double strip layers for a total of 15. A cross section of each detector is shown in Figure 2.

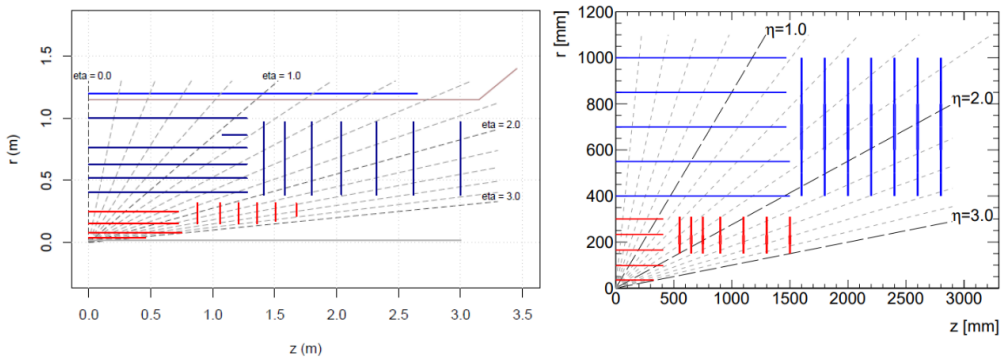


Figure 2: Cross sections of the ATLAS ITk [2] and the generic detector [12]. The design is mirrored in negative z. Red lines represent pixel layers and blue lines strip layers, the brighter blue line in the ATLAS ITk represents the coil of a solenoid magnet. Images, Left: CERN, License: CC-BY-4.0; Right: Gradin et al., fair use.

2.3 Speed requirements

In the HL-LHC, the high rate of events generated, the pileup of 200 [1] events per beam crossing puts stringent time requirements on the Event Filter. As outlined by the relevant Technical Design Report delays of no more than 6-10 μs [3] may be required for the track reconstruction system.

2.4 Bit shifiting

Bit shifting is a powerful tool in specific high-efficiency computing applications, being a method for direct manipulation of the computer's native bits it is generally faster than more high-level methods which can be especially gainful when using FPGAs or other hardware applications [11]. For example 8 divided by 2 in binary is

$$\frac{...01000}{...00010} = ...00100$$

or simply a shift of the bit one step to the right. More operators exist like `AND`, `OR`, `COMPLEMENT` and `XOR` [5]. Another use for bitwise operators is very efficient modulo calculations for powers of 2. The `AND` operator returns 1 if the corresponding bits in both a and b are 1, otherwise 0. So for a mod 2^n set $b = 2^n - 1$. The effect is similar to x mod 10^n in base 10, you keep only the n last digits! For further reading on the subject this Real Python tutorial [6] and this StackOverflow thread [7] are recommended.

3 Materials and Method

To develop an algorithm for correlating local and global coordinates a script implementing said algorithm was written in Python [8] and the ROOT data analysis framework [9]. The data that the script was tested against was a root-file containing a set of 100 000 simulated muon tracks that have generated hits in the generic detector simulated by Gradin et al. [12] using GEANT4 [10].

3.1 Evaluation

To test whether the algorithm is successful or not a set of requirements were made. The algorithm must:

- correlate local and global coordinates
- be able to handle large amounts of data (the dataset used was ∼300 MB total)
- be able to handle multiple strip layers
- be possible to implement in FPGA's in a manner that saves on physical resources (memory and FPGA chips) and computation time

Since Python is a high-level programming language with advanced logic and functions that would be either overly slow, or unavailable [11] in a Hardware Description Language that is used with FPGA's the last requirement was made into a set of constraints that the code must follow to ensure that the algorithm is not overly pythonic:

- Use few steps
- Enable bit manipulation by using factors 2^n whenever possible
- Work in an integer environment since this is faster on an FPGA [11]
- Use parameters, constants and lookup-tables rather than complex arithmetics

3.2 Data

The input data from the simulation that was used to develop the code and test the algorithm was the local x, y, and z coordinates, along with the unique identifiers of the modules called "hash codes", all pictured in Figure 3. To

calculate the error of the method the global X, Y, and Z coordinates, pictured in Figure 4, were also used. Furthermore a limited region (highlighted in red in the figures mentioned above) in layers 5 and 6 in the middle of the most irradiated part of the detector was selected to develop the algorithm on since the high number of hits and limited number of modules would make isolating errors easier.

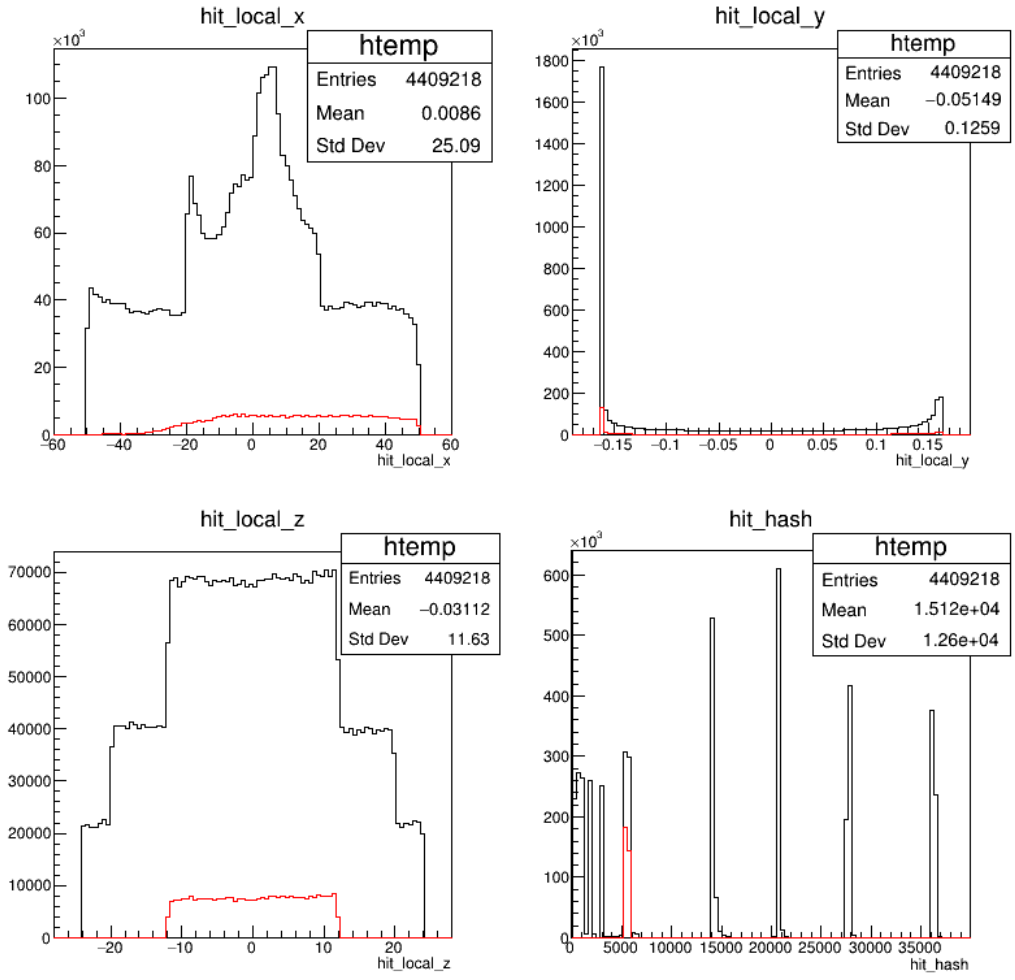


Figure 3: Histograms showing the total distribution of the local coordinates and the hash codes. The red line shows the data selected for use in the Python code and the black is the total.

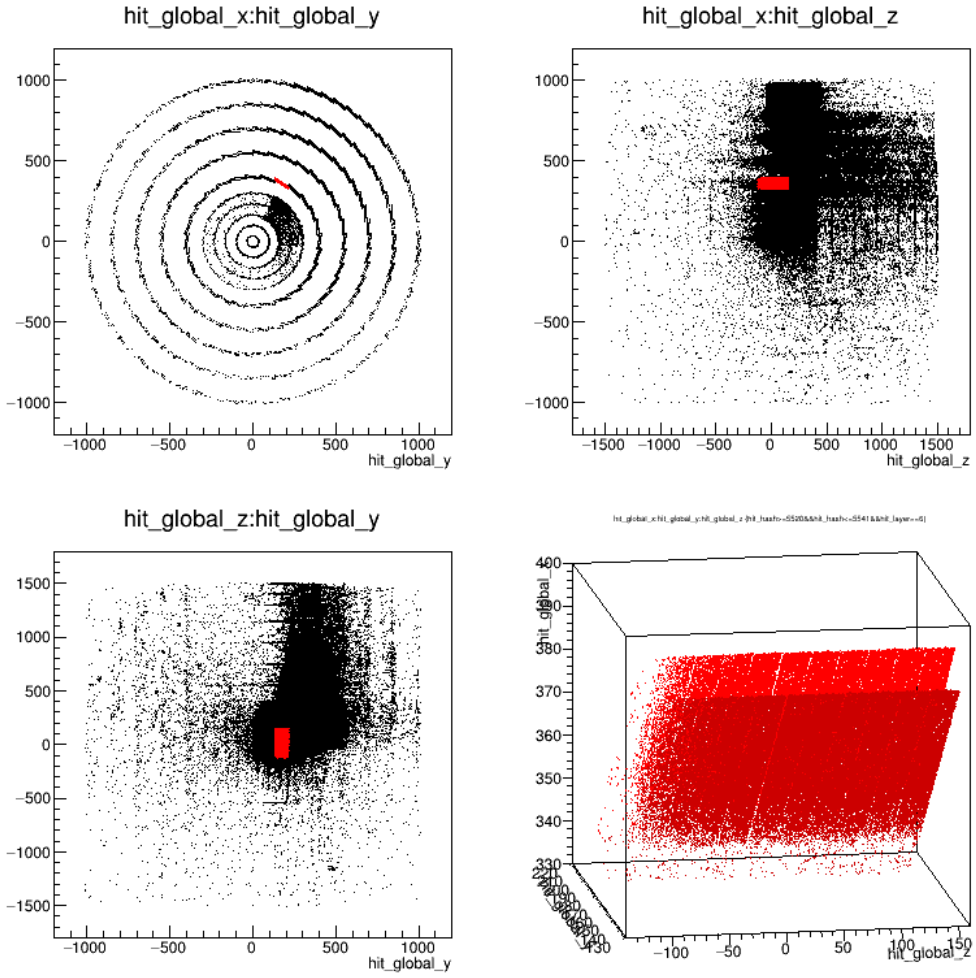


Figure 4: Scatter plot of the global coordinates included in the simulated data, with the selection made in the Python code highlighted in red. Bottom right is a close-up of the selection with the borders between the modules visible and layer 5 given a slightly darker hue.

4 Results

The selection consists of modules with hash codes 5520-5541. Module 5520 is placed at Z=0 and was therefore selected alone to figure out the placement of the local coordinates in the global XY-plane. Through carefully studying the scatter plots in Figure 4 one can see that there are 28 rows of modules in the 5-6-double layer placed at a radius of 400±5 mm, giving an global angle of $\phi = \pi/7$ rad around the Z-axis for the selection. One can also see a small rotation of the module rows from the tangent around the circle and an even smaller rotation in the plane of the modules from the "crossing angle" mentioned in 2.2. Studying the discrepancy when performing the translations allowed the first angle to be determined to be a $-\pi/2$ turn plus a $\pi/18$ rotation along the local z-axis for $\theta = -4\pi/9$ rad and the second to $\gamma = 20$ mrad around the local y-axis.

4.1 The modulo 4 cycle

Extending the selection to contain multiple modules exposed the four-step cycle determining placement in layer 5 or 6, which way the γ-rotation is applied, and the placement of the local origins along the global Z-axis.

1. layer 5, -γ +Z
2. layer 6, +γ +Z
3. layer 5, -γ -Z
4. layer 6, +γ -Z

Further cycles places modules along the Z-axis according to

$$Z = \pm(25N + 13)$$

where N is the number of completed cycles. The row of modules contains 240 modules so $N \leq 59$. The cycle i presented visually in Figure 5.

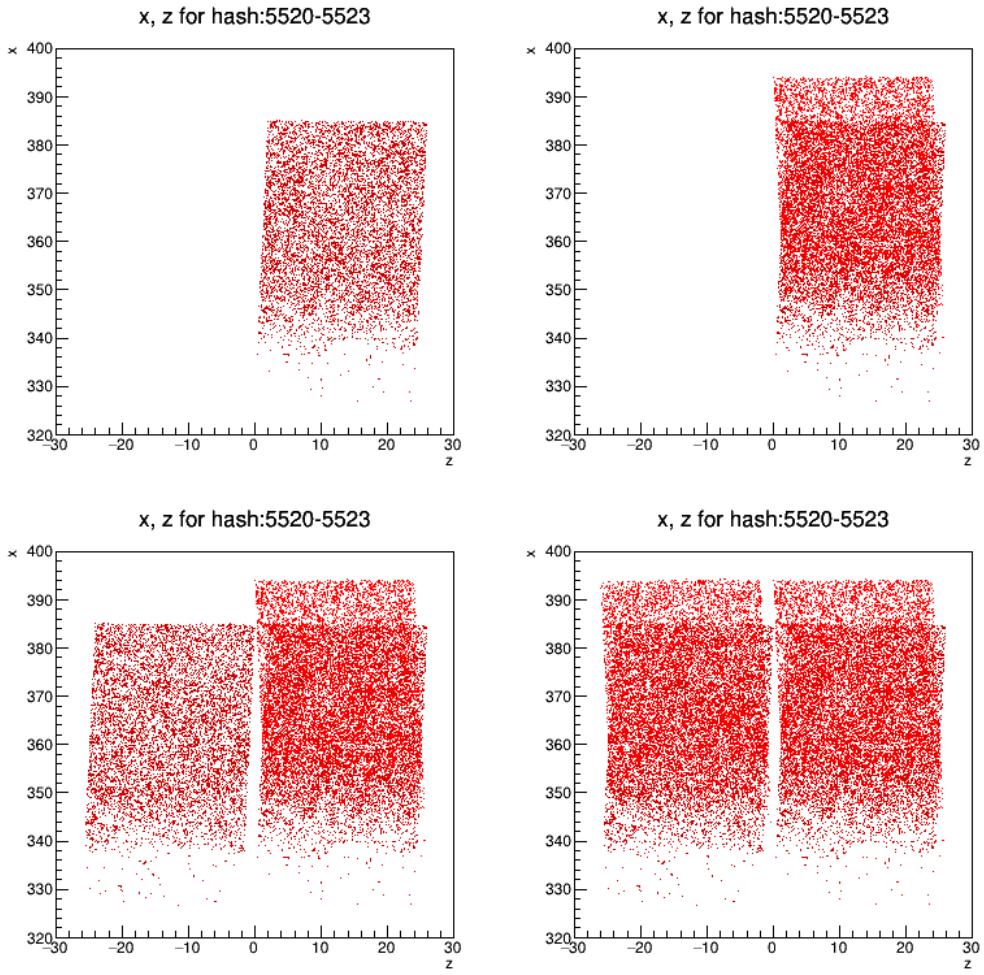


Figure 5: Scatter plot of the global coordinates for hash codes 5520-5523, layer 5 is darker and layer 6 is brighter for visibility. This demonstrates how the modulo 4 (0-3) cycle affects the γ-rotation and placement of the local origin along the global Z-axis, and how the layers are handled. 1's and 3's are in layer 6, and 2's and 3's are in negative Z. The cycle would continue by placing hash 5524 (a 0) to the right of hash 5520, etc.

4.2 Two algorithms

Two algorithms were created, one simply performs the rotations and displacements found at the start of section 4, and one that takes as many shortcuts as possible. The conversion from floating point to integer environments was, similarly to Hettiarachchi et al. [11], done by a scale factor 2^n. Both al-

gorithms were tested for a few n's and with floating-point, to to represent $n \to \infty$.

4.2.1 Rotation algorithm

The following pseudo-code recounts in general the steps taken by the Rotation algorithm. The actual Python code is listed in Appendix A.1.

```
for every hit{
    fetch hash
    if 5520 <= hash <= 5541{
        fetch and scale local xyz and set to temp vars xyz_t
        if hash&3 is 1 or 3{
            invert gamma in rot_Mat_TG
            set R to 395+10}
        else{
            set R to 395}
        create temp vars xyz_t2 = rot_Mat_TG*xyz_t
        Add to x_t2: R
        create global XYZ = rot_Mat_phi*xyz_t2
        if hash&3 >= 2{
            subtract from Z: 25*(hash-5520)>>2 + 13}
        else{
            add to Z: 25*(hash-5520)>>2 + 13}
        output, scaled back down: X,Y,Z
}}
```

Note that in case matrix multiplication is not available the rotations may be done by normal multiplication of each of the temporary coordinate variables with the matrix elements. The latter is the way it is done in the Python code, to be on the safe side. The "`&`" represents the `AND` bit operator mentioned in Section 2.4.

All variables are assumed to be scaled integers and operations to be normalized for the scale. In the Python code this is done by connecting the multiplication operator with a whole number division by the scale factor 2^n. In a more realistic implementation, this would be handled internally, see the vector multiplication kernel in Hettiarachchi's paper for an example [11]. The $\theta\gamma$-matrix, and the $+10$ to the radius are the same for all strip layers, but the radius of 395 is specific to the 5-6-layer and the 5520 base to the selected row in the layer.

4.2.2 Shear algorithm

The shear algorithm uses the small angle approximation to replace the γ rotation, since it is only 20 mrad, even second order terms are discarded. This allows a single rotation around the global Z by the sum of θ and ϕ.

The following pseudo-code recounts in general the steps taken by the Shear algorithm. The actual Python code is listed in Appendix A.2.

```
load constants: sinphi, cosphi, sinPT, cosPT, X_off, Y_off
for every hit{
    fetch hash
    if 5520 <= hash <= 5541{
        fetch and scale local xyz and set to temp vars xyz_t
        create temp var gamma = 0.02
        if hash&3 is 1 or 3{
            create temp var X_L6 = 10*cosphi
            create temp var Y_L6 = 10*sinphi
            invert gamma}
        else{
            create temp var X_L6 = 0
            create temp var Y_L6 = 0}
        create temp var z_off = 25*(hash-5520)>>2 + 13
        if hash&3 >= 2{
            invert z_off}
        create temp var x_t2 = x_t - gamma*z_t
        create temp var z_t2 = z_t + gamma*x_t
        create X = cosPT*x_t2 - sinPT*y_t + X_off + X_L6
        create Y = sinPT*x_t2 + cosPT*y_t + Y_off + Y_L6
        create Z = z_t2 + z_off
        output, scaled back down: X,Y,Z
}}
```

Here the θ-plus-ϕ-rotation is explicitly handled by direct multiplication with constants to save on processing. Note also that the radial displacement is split into x and y factors which are added right after the rotation.

The previous notes on normalization and selection-specific values are also relevant here.

4.3 Discrepancies

The discrepancy between the true globals from the data and the calculated dittos for the two algorithms is plotted in Figure 6. It was tested for $n = 8-13$ and with floating point. The plots illustrate an erratic behaviour in the mean of the errors due the truncation of the 8 trigonometric terms in the rotation matrices in the rotation algorithm. Also shown is how the shear algorithm in a more stable way moves to a minimum, while also achieving decent accuracy already at $n = 8$.

To further illustrate the characteristics of the different algorithms detailed plots of the errors are given in Figures 7, and 8. The former is the result of running the Shear algorithm with $n = 8$ and the latter is from a run of the Rotation algorithm with $n = 11$. The width of the distributions on the right side depend mostly on the error in γ. For X and Y the pitch depends on both θ and ϕ, with a stronger dependence on θ, while the height displacements also depend on both θ and ϕ, but with reversed strength relations.

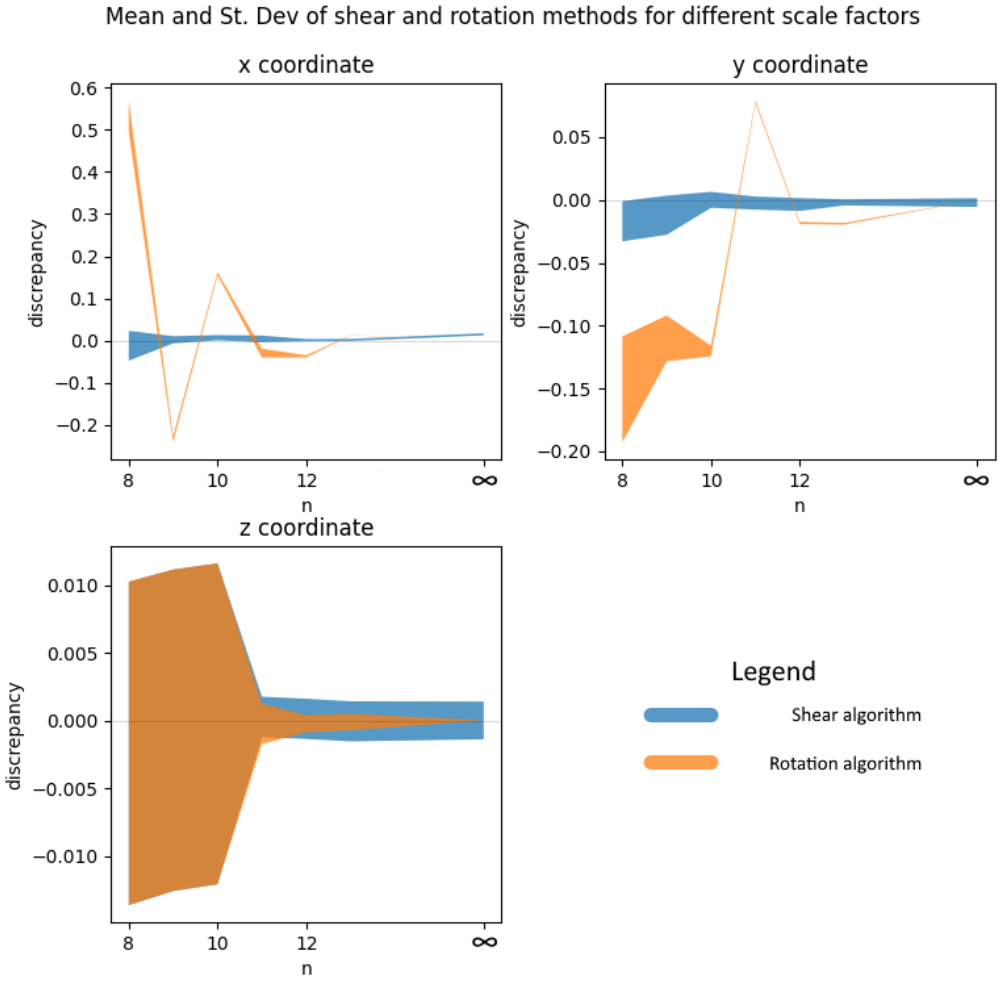


Figure 6: Plots of the mean and standard deviation of the error distributions for $n = 8, 9, 10, 11, 12, 13$ and floating-point for the three coordinates. The height of the line is determined by the mean and the width by the standard deviation. Note that all three of the Rotation-lines go toward 0 in both metrics at floating-point precision, making the lines difficult to see. The initially chaotic behaviour of the mean paired with the rapid reduction of the width of the distribution from the Rotation algorithm is contrasted with the stable but limited convergence of the Shear algorithm.

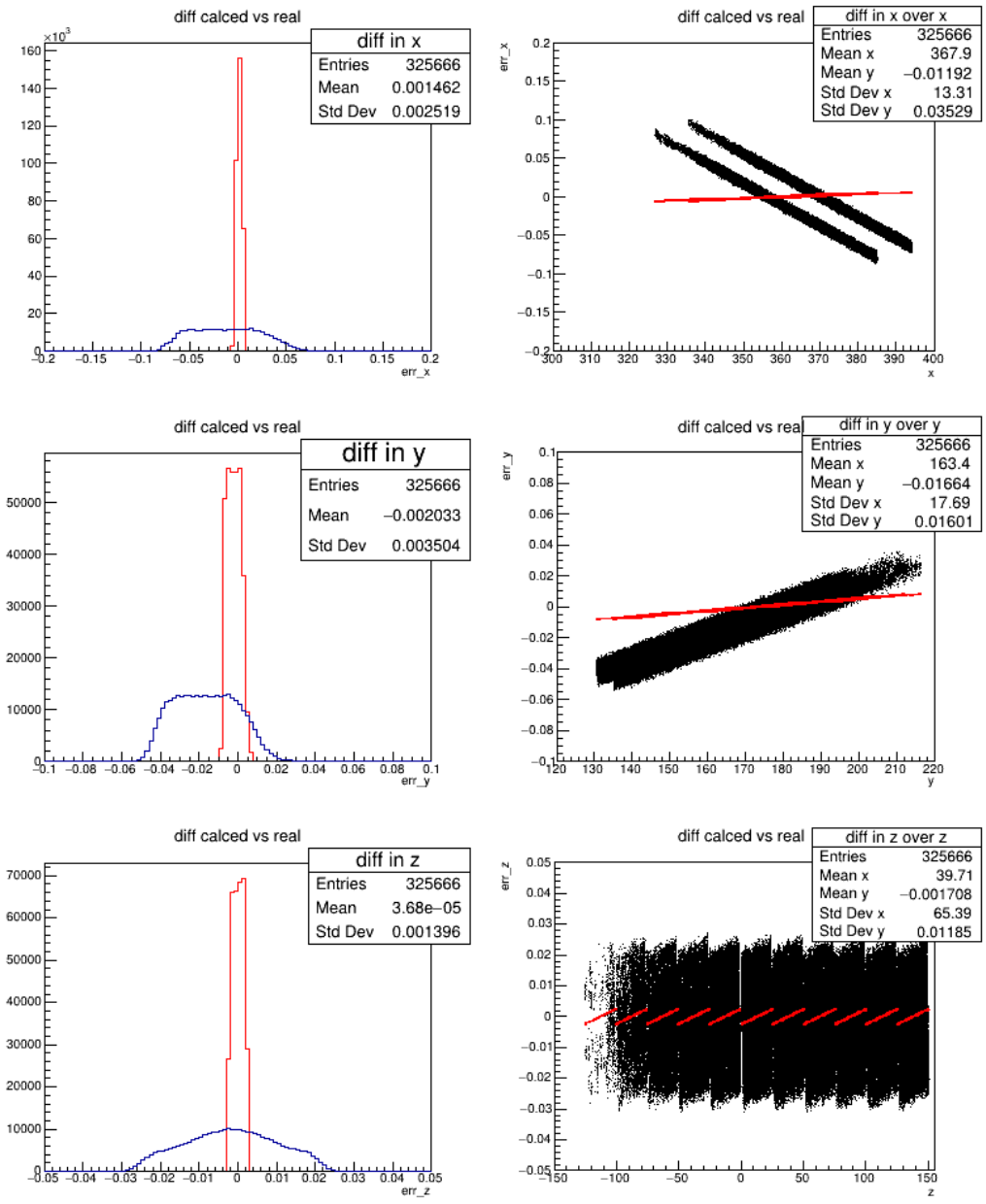


Figure 7: Detail of errors from running the Shear algorithm with $n = 8$. On the left is a histogram to show the distribution and on the right is a scatter plot where the error's dependence on the own coordinate is examined. The floating-point, or maximum precision, solution is shown in red. The stats on the left are for the floating-point distribution.

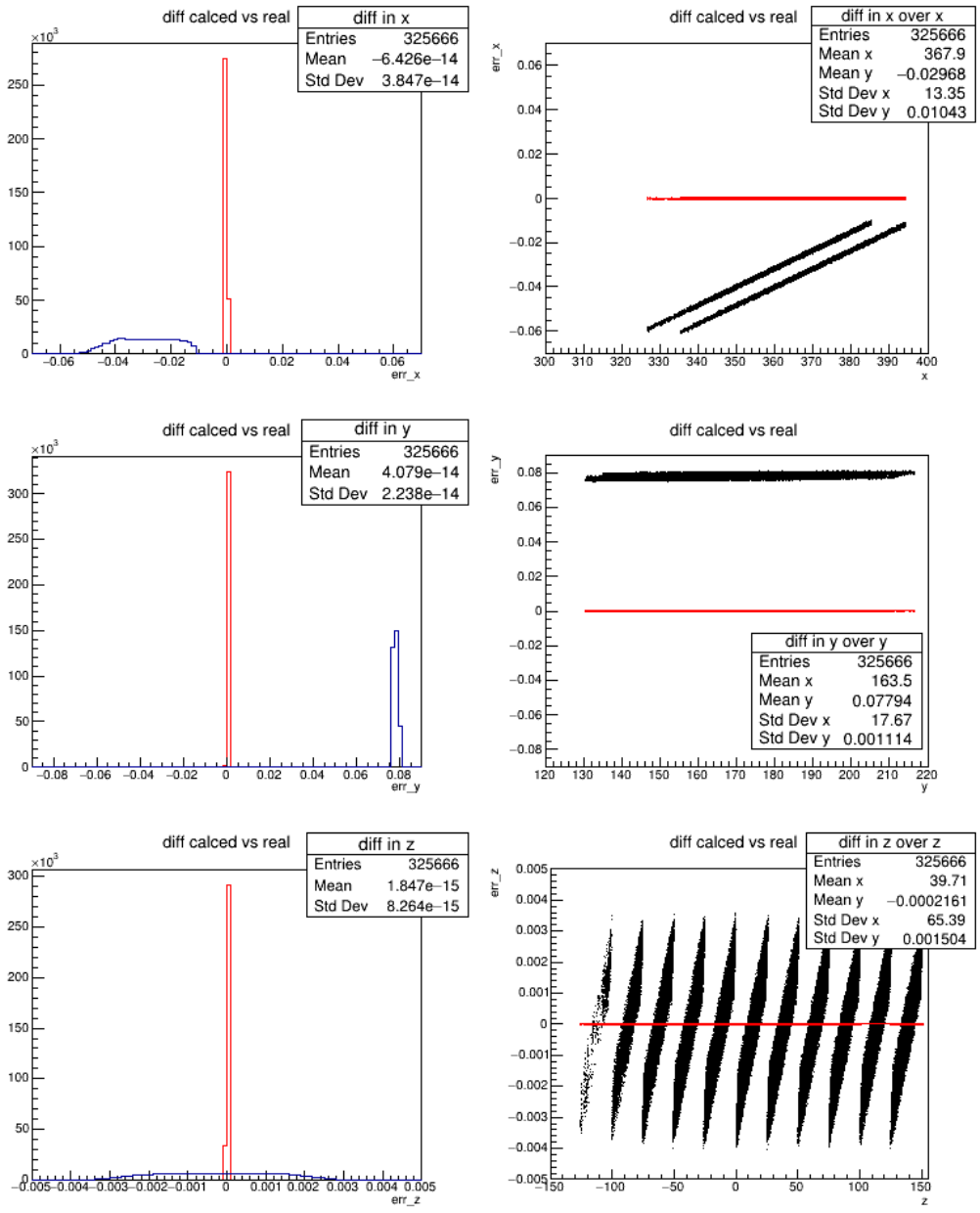


Figure 8: Detail of errors from running the Rotation algorithm with $n = 11$. On the left is a histogram to show the distribution and on the right is a scatter plot where the error's dependence on the own coordinate is examined. The floating-point, or maximum precision, solution is shown in red. The stats on the left are for the floating-point distribution.

5 Discussion and Outlook

The goal of this thesis was to develop a coordinate conversion algorithm for detectors that could be implemented in FPGAs to enable the use of the Hough transform in particle path reconstruction.

The requirements and limitations outlined in section 3.1 are at least partially fulfilled by the Shear algorithm that was developed. The biggest limitation pertains to the expansion of the selection into multiple rows and multiple double strip layers, since that part is completely hard-coded in the current implementation. The first step one would take in further developing the algorithm would be to generalise this part. Though that would come with its own challenges, among other things the number of rows is not the same in every layer.

Both versions of the algorithm contain trigonometric functions which are non-trivial to implement in an FPGA. A common way to implement such functions is look-up tables, as shown by Hettiarachchi et al. [11], which should be sufficient here too. However it might be prudent to limit the use as demonstrated by the chaotic behaviour of the rotation algorithm in Figure 6, which comes from the truncation of the 6 trigonometric variables.

If the Hough transform were to be set up to accept cylindrical coordinates it might be worth to explore remaking the algorithm in a cylindrical environment, since it would greatly simplify the rotations.

From this thesis it seems that the Shear algorithm has the most advantages, since the small angle approximation simplifies the rotations, providing stability even with small scale factors and only loses a small amount of precision. However if higher precision is needed and larger scale factors isn't an issue, the Rotation algorithm might be the only option. Since it contains 3 elementary rotations and a displacement using a fast algorithm for rotations is a priority. In this case the CORDIC [14] algorithm might be worth investigating in a future study.

Upgrading the ATLAS detector is a large undertaking. The potential savings in resources and work hours from avoiding the custom AM-ASICs, which have to be manufactured in-house and are also inflexible with regards to the detector structure [12], motivates developing the Hough implementation further.

6 Conclusions

Given that the global coordinates used in the Hough transform should be cartesian and limits to the size of the scale factor the Shear algorithm is a promising attempt at a conversion algorithm, however it still requires an extension to handle all layers around the whole detector. The Rotation algorithm is possible to use, but only with large scale factors, since it is very sensitive to the truncation error. It also has the same need for an extension into more layers.

References

[1] I. Béjar Alonso et al. (Eds.) *High-Luminosity Large Hadron Collider (HL-LHC): Technical design report*, CERN Yellow Reports: Monographs, CERN-2020-010. CERN, Geneva, 2020, https://doi.org/10.23731/CYRM-2020-0010

[2] The ATLAS Collaboration, Technical Design Report for the ATLAS inner tracker strip detector, 2017. Fetched May 2021 https://cds.cern.ch/record/2257755/files/ATLAS-TDR-025.pdf

[3] The ATLAS Collaboration, Technical Design Report for the Phase-II Upgrade of the ATLAS Trigger and Data Acquisition System, 2018. Fetched May 2021 https://cds.cern.ch/record/2285584/files/ATLAS-TDR-029.pdf

[4] Richard O. Duda, Peter E. Hart, *Use of the Hough Transformation to Detect Lines and Curves in Pictures*, Comm. ACM. vol 15 1972. Fetched May 2021 http://www.ai.sri.com/pubs/files/tn036-duda71.pdf

[5] Wiki contributors *BitwiseOperators* The Python Wiki, last edit Jul 2013. Fetched May 2021 https://wiki.python.org/moin/BitwiseOperators

[6] Bartosz Zaczyński *Python bitwise operators* Real Python Tutorials, Dec 2020. Fetched May 2021 https://realpython.com/python-bitwise-operators/

[7] Chrisapotek *What is the rationale behind (x % 64) == (x & 63)? [duplicate]* StackOverflow, last edited May 2017. Fetched May 2021 https://stackoverflow.com/questions/13784790/what-is-the-rationale-behind-x-64-x-63

[8] Python Homepage https://www.python.org

[9] ROOT Homepage https://root.cern/

[10] GEANT4 homepage https://geant4.web.cern.ch/node/1

[11] Don Lahiru Nirmal Hettiarachchi, Venkata Salini Priyamvada Davuluru and Eric J. Balster. *Integer vs. Floating-Point Processing on Modern FPGA Technology*, 10th Annual Computing and Communication Workshop and Conference (CCWC), 2020, pp. 0606-0612. doi: 10.1109/CCWC47524.2020.9031118, https://ieeexplore.ieee.org/document/9031118

[12] Joakim Gradin, Mikael Mårtensson, and Richard Brenner. *Comparison of two hardware-based hit filtering methods for trackers in high-pileup environments*, Journal of Instrumentation vol 13. IOP Publishing 2018. DOI: 10.1088/1748-0221/13/04/P04019 `https://arxiv.org/abs/1709.01034`

[13] Mikael Mårtensson, Max Isacson, Hampus Hahne, Rebeca Gonzalez Suarez and Richard Brenner. *To catch a long-lived particle: hit selection towards a regional hardware track trigger implementation*, Journal of Instrumentation vol. 14. IOP Publishing 2019. DOI: 10.1088/1748-0221/14/11/p11009 `https://arxiv.org/abs/1907.09846`

[14] Xiao-Gang Jiang, Jian-Yang Zhou, Jiang-Hong Shi and Hui-Huang Chen, *FPGA implementation of image rotation using modified compensated CORDIC*, 2005 6th International Conference on ASIC, 2005, pp. 752-756, doi: 10.1109/ICASIC.2005.1611424. `https://ieeexplore.ieee.org/document/1611424`

A Code

This Appendix contains listings of the Python scripts that were made for this thesis. A.1 contains the script for the rotation algorithm. A.2 contains the script for the shear algorithm.

A.1 rotation.py

```
# import libraries
import sys
from typing import SupportsIndex
from ROOT import TCanvas, TPad, TFile, TPaveText, TBrowser
from ROOT import gBenchmark, gStyle, gROOT, TH1D, TH1I, TH3D,
    TH2D
from time import time, sleep
from math import atan, pi, tan, sin, cos

# create a canvas and histogram
c1 = TCanvas('c1','The Ntuple canvas',200,10,800,1000)
c2 = TCanvas('c2','The Ntuple canvas',1000,10,800,800)

# Scaling parameter for integer solution
n = 11
mlt = 2**n
print("scaling by: ",mlt)
#                    256 512    1024 2048   4096     8192
errhxu = 0.07     #   0.8 -0.1   0.2  0      -0.025   0.015
errhxl = -0.07    #   0.4 -0.3   0.1 -0.07   -0.045   0.011
errhyu = 0.09    #  0     0    -0.1   0.085 -0.015 -0.015
errhyl = -0.09    # -0.3 -0.3 -0.16 0.07   -0.022 -0.022
errhz = 0.005     #   0.05 0.05 0.05 0.005   0.002   0.002

#               name,  title,                    nbinsx,xl ,
   xu,nby,yl, yu,nbz,zl,zu
h_calcxy = TH2D("x, y","Calculated global coordinates;y;x"
    ,1000,120,220,1000,300,400)
h_calcxz = TH2D("x, z","Calculated global coordinates;z;x"
    ,3050,-150,155,1000,300,400)
h_calcyz = TH2D("z, y","Calculated global coordinates;y;z"
    ,1000,120,220,3050,-150,155)
h_calcxy.SetStats(0)
h_calcxz.SetStats(0)
h_calcyz.SetStats(0)
h_diffxx = TH2D("diff in x over x","diff calced vs real;x;
   err_x",1000,300,400,300,errhxl,errhxu)
h_diffyy = TH2D("diff in y over y","diff calced vs real;y;
   err_y",1000,120,220,300,errhyl,errhyu)
```

```
h_diffzz = TH2D("diff in z over z","diff calced vs real;z;
    err_z",3050,-150,155,300,-errhz,errhz)
h_diffx = TH1D("I diff in x","diff calced vs real;err_x",100,
    errhxl,errhxu)
h_diffy = TH1D("I diff in y","diff calced vs real;err_y",100,
    errhyl,errhyu)
h_diffz = TH1D("I diff in z","diff calced vs real;err_z"
    ,100,-errhz,errhz)

h_exactxy = TH2D("f: x, y","float globs from locs;y;x"
    ,1000,120,220,1000,300,400)
h_exactxz = TH2D("f: x, z","float globs from locs;z;x"
    ,3050,-150,155,1000,300,400)
h_exactyz = TH2D("f: y, z","float globs from locs;y;z"
    ,1000,120,220,3050,-150,155)
h_exactxy.SetStats(0)
h_exactxz.SetStats(0)
h_exactyz.SetStats(0)
h_exdiffxx = TH2D("f: diff in x over x","float from real;x;
    err_x",1000,300,400,300,errhxl,errhxu)
h_exdiffyy = TH2D("f: diff in y over y","float from real;y;
    err_y",1000,120,220,300,errhyl,errhyu)
h_exdiffzz = TH2D("f: diff in z over z","float globs from
    real globs;z;err_z",3050,-150,155,300,-errhz,errhz)
h_exdiffx = TH1D("diff in x","diff calced vs real;err_x",100,
    errhxl,errhxu)
h_exdiffy = TH1D("diff in y","diff calced vs real;err_y",100,
    errhyl,errhyu)
h_exdiffz = TH1D("diff in z","diff calced vs real;err_z"
    ,100,-errhz,errhz)

#open file
file = TFile("./muons_eta0.1-0.3_run0.root",'read') # modify
    as required to reach the file on your specific system
tree = file.Get("tracking")

#b = TBrowser() #useful for looking at available parameters
    in the data

#hash code (Module ID) selection
hnl = 5520
hnu = 5541

#precalculated parameters
ang_z1 = -pi*4/9# pi(-1/2+0.5/9) #rotation angle in local xy
    plane
ang_y1 = 0.02 # rotation angle in local xz plane
ang_z2 = pi/7 #rotation angle in global xy plane
```

```
65 # The above angle is dependen both on the specific row in a
       layer and the number of rows in the layer
66 # This will at least require mod 240, which is possible using
       bit shifts. It will also likely require
67 # some manner of lookup table for hash:layers unless layers
       are knownsince total number of hashes per layer is not
68 # constant.
69
70
71 #exact solution values
72 sinz1 = sin(ang_z1)
73 cosz1 = cos(ang_z1)
74 siny0 = sin(ang_y1)
75 cosy0 = cos(ang_y1)
76 sinz2 = sin(ang_z2)
77 cosz2 = cos(ang_z2)
78
79
80 #integer solution values
81 isinz1 = int(round(mlt*sin(ang_z1)))
82 icosz1 = int(round(mlt*cos(ang_z1)))
83 isiny0 = int(round(mlt*sin(ang_y1)))
84 icosy0 = int(round(mlt*cos(ang_y1)))
85 isinz2 = int(round(mlt*sin(ang_z2)))
86 icosz2 = int(round(mlt*cos(ang_z2)))
87
88
89 # some measurement variables
90 lpcnt = 0
91 loop_start = time()
92 print("loop start")
93 #loop over ntuple
94 for entry in tree:
95     # a loop limiter for quick code testing
96     lpcnt += 1
97     # if lpcnt == 10000:
98     #     break
99
100    # Filling using matching elements
101    for i,hsh in enumerate(tree.hit_hash):
102        if hnl <= hsh <= hnu:
103            #handling of the mod 4 hash code cycle
104            hsh -= hnl
105            big = hsh>>2
106            sml = hsh&3
107
108
109            # Fetch local coords
110            xloc_t1 = int(round(mlt*tree.hit_local_x[i]))
```

```
111             yloc_t1 = int(round(mlt*tree.hit_local_y[i]))
112             zloc_t1 = int(round(mlt*tree.hit_local_z[i]))
113 
114             #reset these params
115             isiny1 = isiny0
116             icosy1 = icosy0
117             #modify according to cycle
118             if sml in (1,3):
119                 isiny1 *= -1
120                 r_off = mlt*405
121             else:
122                 r_off = mlt*395
123 
124 
125             # note that all multiplications must be shifted
    down by n to keep the scaling
126             # this would be handled internally in the
    multiplication kernel in openCL,
127             # but has to be done explicitly with whole number
     division here.
128 
129             # Local rotations are same for all modules
130             # Prepare combined local rotation (split for
    readability)
131             r1 = icosy1*xloc_t1//mlt
132             r2 = isiny1*zloc_t1//mlt
133 
134             xloc = icosz1*(r1-r2)//mlt - isinz1*yloc_t1//mlt
135             yloc = isinz1*(r1-r2)//mlt + icosz1*yloc_t1//mlt
136             zloc = isiny1*xloc_t1//mlt + icosy1*zloc_t1//mlt
137 
138             z_off = mlt*(25*big + 13)
139             if sml >= 2:
140                 z_off *= -1 #easy bit shift
141 
142             #following global offset and rotation will be
    different for different modules
143             xloc += r_off
144 
145             # The rotation here is dependent both on the
    specific row of modules and the number of rows in the
    specific layer
146             # The most efficient way to implement will
    probably be some manner of lookup tables.
147             x = icosz2*xloc//mlt - isinz2*yloc//mlt
148             y = isinz2*xloc//mlt + icosz2*yloc//mlt
149             z = z_off + zloc
150 
151 
```

```
            # Scale back down for plots
            x /= mlt
            y /= mlt
            z /= mlt

            #compare with true values
            delx = x - tree.hit_global_x[i]
            dely = y - tree.hit_global_y[i]
            delz = z - tree.hit_global_z[i]

            #fill histograms
            h_calcxz.Fill(z,x)
            h_calcxy.Fill(y,x)
            h_calcyz.Fill(y,z)

            h_diffx.Fill(delx)
            h_diffy.Fill(dely)
            h_diffz.Fill(delz)

            h_diffxx.Fill(x,delx)
            h_diffyy.Fill(y,dely)
            h_diffzz.Fill(z,delz)



            #Repeat for exact solution
            # Fetch local coords
            xloc_t1 = tree.hit_local_x[i]
            yloc_t1 = tree.hit_local_y[i]
            zloc_t1 = tree.hit_local_z[i]

            #reset these params
            #ang_y1 = 0.02
            siny1 = siny0
            cosy1 = cosy0
            r_offex = 395
            #modify according to cycle
            if sml in (1,3):
                #ang_y1 *=-1
                siny1 *= -1
                r_offex += 10

            # Local rotations are same for all modules
            # Do combined local rotation
            xloc = cosz1*(cosy1*xloc_t1 - siny1*zloc_t1) -
sinz1*yloc_t1
            yloc = sinz1*(cosy1*xloc_t1 - siny1*zloc_t1) +
cosz1*yloc_t1
            zloc = siny1*xloc_t1 + cosy1*zloc_t1
```

```
199
200             z_off = 25*big + 13
201             if sml >= 2:
202                 z_off *= -1 #easy bit shift
203
204             #following global offset and rotation will be
    different for different modules
205             xloc += r_offex
206
207             # The rotation here is dependent both on the
    specific row of modules and the number of rows in the
    specific layer
208             # The most efficient way to implement will
    probably be some manner of lookup tables.
209             x = cosz2*xloc - sinz2*yloc
210             y = sinz2*xloc + cosz2*yloc
211             z = z_off + zloc
212
213             # print(x,y,z)
214
215             delx = x - tree.hit_global_x[i]
216             dely = y - tree.hit_global_y[i]
217             delz = z - tree.hit_global_z[i]
218
219
220             h_exactxy.Fill(y,x)
221             h_exactxz.Fill(z,x)
222             h_exactyz.Fill(y,z)
223
224             h_exdiffx.Fill(delx)
225             h_exdiffy.Fill(dely)
226             h_exdiffz.Fill(delz)
227
228             h_exdiffxx.Fill(x,delx)
229             h_exdiffyy.Fill(y,dely)
230             h_exdiffzz.Fill(z,delz)
231
232
233
234
235
236
237 loop_done = time()
238 print(f"loop time: {loop_done-loop_start}")
239 print(lpcnt)
240
241 # Plot histogram
242 # Divide canvas for two plots on same canvas
243 c1.Divide(2,3)
```

```
244
245 c1.cd(1)
246 h_exdiffx.SetLineColor(2)
247 h_exdiffx.Draw()
248 h_diffx.Draw("SAME")
249
250 c1.cd(2)
251 h_diffxx.SetMarkerColor(1)
252 h_diffxx.Draw()
253 h_exdiffxx.SetMarkerColor(2)
254 h_exdiffxx.Draw("SAME")
255
256 c1.cd(3)
257 h_exdiffy.SetLineColor(2)
258 h_exdiffy.Draw()
259 h_diffy.Draw("SAME")
260
261 c1.cd(4)
262 h_diffyy.SetMarkerColor(1)
263 h_diffyy.Draw()
264 h_exdiffyy.SetMarkerColor(2)
265 h_exdiffyy.Draw("SAME")
266
267 c1.cd(5)
268 h_exdiffz.SetLineColor(2)
269 h_exdiffz.Draw()
270 h_diffz.Draw("SAME")
271
272 c1.cd(6)
273 h_diffxx.SetMarkerColor(1)
274 h_diffzz.Draw()
275 h_exdiffzz.SetMarkerColor(2)
276 h_exdiffzz.Draw("SAME")
277
278 c1.Update()
279 # this statement is to allow to inspects plots before code
      finishes
280
281
282 c2.Divide(2,2)
283
284 c2.cd(1)
285 #h_exactxy.Draw()
286 h_calcxy.Draw()
287
288 c2.cd(2)
289 #h_exactxz.Draw()
290 h_calcxz.Draw()
291
```

```
292 c2.cd(3)
293 #h_exactyz.Draw()
294 h_calcyz.Draw()
295
296 c2.cd(4)
297 #h_diffzz.Draw("")
298
299 c2.Update()
300
301
302
303 # the following is code to show how hashes are mapped in the
      z-direction,
304 # turns out they're mapped from 0 outward, alternating
      between negatives and positives
305
306 # c1.cd(2)
307 # opts = f"hit_hash>={hnl}&&hit_hash<={hnu}"
308 # hist=tree.Draw("hit_hash",opts)
309
310 # for i in range(hnl,hnu,2):
311 #     opts2 = f"hit_hash>={i-1}&&hit_hash<={i}"
312 #     c1.cd(1)
313 #     hist=tree.Draw("hit_global_x:hit_global_z",opts2)
314 #     c1.Update()
315 #     sleep(0.5)
316
317 code_done = time()
318 print(f"time to execute: {code_done-loop_start}")
319 r=input('Please press enter to continue.')
```

A.2 shear.py

```
# import libraries
import sys
from typing import SupportsIndex
from ROOT import TCanvas, TPad, TFile, TPaveText, TBrowser
from ROOT import gBenchmark, gStyle, gROOT, TH1D, TH1I, TH3D,
    TH2D
from time import time, sleep
from math import atan, pi, tan, sin, cos

# create a canvas and histogram
c1 = TCanvas('c1','The Ntuple canvas',200,10,800,1000)
c2 = TCanvas('c2','The Ntuple canvas',1000,10,800,800)

# Scaling parameter for integer solution
n = 8
mlt = 2**n
print("scaling by: ",mlt)

#                     256    512    1024   2048    4096    8192
errhxu = 0.2      #   0.2    0.04   0.03   0.02    0.01    0.01
errhxl = -0.2     #  -0.2   -0.04  -0.03  -0.02    0.01   -0.01
errhyu = 0.1      #   0.1    0.04   0.03   0.02    0.015   0.01
errhyl = -0.1     #  -0.1   -0.06  -0.03  -0.02   -0.015  -0.01
errhz = 0.05      #   0.05   0.05   0.03   0.004   0.004   0.004

#               name,  title,                    nbinsx,xl ,
   xu,nby,yl, yu,nbz,zl,zu
h_calcxy = TH2D("x, y","Calculated global coordinates;y;x"
    ,1000,120,220,1000,300,400)
h_calcxz = TH2D("x, z","Calculated global coordinates;z;x"
    ,3050,-150,155,1000,300,400)
h_calcyz = TH2D("z, y","Calculated global coordinates;y;z"
    ,1000,120,220,3050,-150,155)
h_calcxy.SetStats(0)
h_calcxz.SetStats(0)
h_calcyz.SetStats(0)
h_diffxx = TH2D("diff in x over x","diff calced vs real;x;
    err_x",1000,300,400,300,errhxl,errhxu)
h_diffyy = TH2D("diff in y over y","diff calced vs real;y;
    err_y",1000,120,220,300,errhyl,errhyu)
h_diffzz = TH2D("diff in z over z","diff calced vs real;z;
    err_z",3050,-150,155,300,-errhz,errhz)
h_diffx = TH1D("I diff in x","diff calced vs real;err_x",100,
    errhxl,errhxu)
h_diffy = TH1D("I diff in y","diff calced vs real;err_y",100,
    errhyl,errhyu)
h_diffz = TH1D("I diff in z","diff calced vs real;err_z"
```

```
    ,100,-errhz,errhz)

h_exactxy = TH2D("f: x, y","float globs from locs;y;x"
    ,1000,120,220,1000,300,400)
h_exactxz = TH2D("f: x, z","float globs from locs;z;x"
    ,3050,-150,155,1000,300,400)
h_exactyz = TH2D("f: y, z","float globs from locs;y;z"
    ,1000,120,220,3050,-150,155)
h_exactxy.SetStats(0)
h_exactxz.SetStats(0)
h_exactyz.SetStats(0)
h_exdiffxx = TH2D("f: diff in x over x","float from real;x;
    err_x",1000,300,400,300,errhxl,errhxu)
h_exdiffyy = TH2D("f: diff in y over y","float from real;y;
    err_y",1000,120,220,300,errhyl,errhyu)
h_exdiffzz = TH2D("f: diff in z over z","float globs from
    real globs;z;err_z",3050,-150,155,300,-errhz,errhz)
h_exdiffx = TH1D("diff in x","diff calced vs real;err_x",100,
    errhxl,errhxu)
h_exdiffy = TH1D("diff in y","diff calced vs real;err_y",100,
    errhyl,errhyu)
h_exdiffz = TH1D("diff in z","diff calced vs real;err_z"
    ,100,-errhz,errhz)
#open file
file = TFile("./muons_eta0.1-0.3_run0.root",'read') # modify
    as required to reach the file on your specific system
tree = file.Get("tracking")

#b = TBrowser() #useful for looking at available parameters
    in the data

#hash code (Module ID) selection
hnl = 5520
hnu = 5541

#precalculated parameters

#radial displacement : 395 (+10 for layer 6)

#exact solution values
phi = pi/7
phiThet = -pi*19/63#-0.9474933115904356

sinex = sin(phiThet)#-0.8119548525165163
cosex = cos(phiThet)#0.583720239048538
sinphi = sin(phi)
cosphi = cos(phi)
x_offex = 395*cosphi
y_offex = 395*sinphi
```

```
75
76 # integer solution values:
77 sine   = int(round(mlt*sinex))
78 cose   = int(round(mlt*cosex))
79 icosphi = int(round(mlt*cosphi))
80 isinphi = int(round(mlt*sinphi))
81 x_off  = int(round(mlt*x_offex))
82 y_off  = int(round(mlt*y_offex))
83
84
85
86
87 # some measurement variables
88 lpcnt = 0
89 loop_start = time()
90 print("loop start")
91 #loop over ntuple
92 for entry in tree:
93     # a loop limiter for quick code testing
94     lpcnt += 1
95     # if lpcnt == 10000:
96     #     break
97
98     # Filling using matching elements
99     for i,hsh in enumerate(tree.hit_hash):
100         if hnl <= hsh <= hnu:
101             #handling of the mod 4 hash code cycle
102             hsh -= hnl
103             big = hsh>>2
104             sml = hsh&3
105             R_off = 395
106
107             # Do conversion steps for integer solution
108             x_t = int(round(mlt*tree.hit_local_x[i]))
109             yloc = int(round(mlt*tree.hit_local_y[i]))
110             z_t = int(round(mlt*tree.hit_local_z[i]))
111
112
113
114             z_off = mlt*(25*big + 13)
115             if sml >= 2:
116                 z_off *= -1 #easy bit shift
117
118             gamma = int(round(mlt*0.02))
119             if sml in (1,3):
120                 x_L6 = 10*icosphi
121                 y_L6 = 10*isinphi
122                 gamma *= -1
123             else:
```

```
                x_L6 = 0
                y_L6 = 0

            xloc = x_t - gamma*z_t//mlt
            zloc = z_t + gamma*x_t//mlt

            x = cose*xloc//mlt - sine*yloc//mlt + x_off +
    x_L6
            y = sine*xloc//mlt + cose*yloc//mlt + y_off +
    y_L6
            z = zloc + z_off


            #Scale down for plots:
            x /= mlt
            y /= mlt
            z /= mlt

            delx = x - tree.hit_global_x[i]
            dely = y - tree.hit_global_y[i]
            delz = z - tree.hit_global_z[i]


            h_calcxz.Fill(z,x)
            h_calcxy.Fill(y,x)
            h_calcyz.Fill(y,z)

            h_diffx.Fill(delx)
            h_diffy.Fill(dely)
            h_diffz.Fill(delz)

            h_diffxx.Fill(x,delx)
            h_diffyy.Fill(y,dely)
            h_diffzz.Fill(z,delz)




            #Now repeat steps for exact solution

            x_t1 = tree.hit_local_x[i]
            y_t1 = tree.hit_local_y[i]
            z_t1 = tree.hit_local_z[i]
            # [cosa  0 sina] [x]  x' = cosa*x + sina*z
          x' = (1-a^2)*x + a*z ~= x + a*z
            # [0     1    0]*[y]  y' = y
    SAA:   y' = y
            # [-sina 0 cosa] [z]  z' = -sina*x + cosa*z
          z' = -a*x + (1-a^2)z ~= -a*x + z
```

```
            # x_off = 355.875
            # y_off = 171.4
            if sml in (1,3):
                gamma = -0.02
            else:
                gamma = 0.02

            xloc = x_t1 - gamma*z_t1
            yloc = y_t1
            zloc = z_t1 + gamma*x_t1


            z_off = 25*big + 13

            if sml >= 2:
                z_off *= -1 #easy bit shift

            z_ang = 0.02*xloc

            x = cosex*xloc - sinex*yloc + x_offex
            y = sinex*xloc + cosex*yloc + y_offex
            z = zloc + z_off

            if sml in (1,3):
                x += 10*cosphi
                y += 10*sinphi
                z_ang *= -1
            #z += z_ang

            delx = x - tree.hit_global_x[i]
            dely = y - tree.hit_global_y[i]
            delz = z - tree.hit_global_z[i]


            h_exactxy.Fill(x,y)
            h_exactxz.Fill(x,z)
            h_exactyz.Fill(y,z)

            h_exdiffx.Fill(delx)
            h_exdiffy.Fill(dely)
            h_exdiffz.Fill(delz)

            h_exdiffxx.Fill(x,delx)
            h_exdiffyy.Fill(y,dely)
            h_exdiffzz.Fill(z,delz)
```

```
217
218
219
220 loop_done = time()
221 print(f"loop time: {loop_done-loop_start}")
222 print(lpcnt)
223
224 # Plot histogram
225 # Divide canvas for two plots on same canvas
226 c1.Divide(2,3)
227
228 c1.cd(1)
229 h_exdiffx.SetLineColor(2)
230 h_exdiffx.Draw()
231 h_diffx.Draw("SAME")
232
233 c1.cd(2)
234 h_diffxx.SetMarkerColor(1)
235 h_diffxx.Draw()
236 h_exdiffxx.SetMarkerColor(2)
237 h_exdiffxx.Draw("SAME")
238
239 c1.cd(3)
240 h_exdiffy.SetLineColor(2)
241 h_exdiffy.Draw()
242 h_diffy.Draw("SAME")
243
244 c1.cd(4)
245 h_diffyy.SetMarkerColor(1)
246 h_diffyy.Draw()
247 h_exdiffyy.SetMarkerColor(2)
248 h_exdiffyy.Draw("SAME")
249
250 c1.cd(5)
251 h_exdiffz.SetLineColor(2)
252 h_exdiffz.Draw()
253 h_diffz.Draw("SAME")
254
255 c1.cd(6)
256 h_diffxx.SetMarkerColor(1)
257 h_diffzz.Draw()
258 h_exdiffzz.SetMarkerColor(2)
259 h_exdiffzz.Draw("SAME")
260
261 c1.Update()
262 # this statement is to allow to inspects plots before code
      finishes
263
264
```

```
265 c2.Divide(2,2)
266
267 c2.cd(1)
268 #h_exactxy.Draw()
269 h_calcxy.Draw()
270
271 c2.cd(2)
272 #h_exactxz.Draw()
273 h_calcxz.Draw()
274
275 c2.cd(3)
276 #h_exactyz.Draw()
277 h_calcyz.Draw()
278
279 c2.cd(4)
280 #h_diffzz.Draw("")
281
282 c2.Update()
283
284
285
286 # the following is code to show how hashes are mapped in the
      z-direction,
287 # turns out they're mapped from 0 outward, alternating
      between negatives and positives
288
289 # c1.cd(2)
290 # opts = f"hit_hash>={hnl}&&hit_hash<={hnu}"
291 # hist=tree.Draw("hit_hash",opts)
292
293 # for i in range(hnl,hnu,2):
294 #     opts2 = f"hit_hash>={i-1}&&hit_hash<={i}"
295 #     c1.cd(1)
296 #     hist=tree.Draw("hit_global_x:hit_global_z",opts2)
297 #     c1.Update()
298 #     sleep(0.5)
299
300 code_done = time()
301 print(f"time to execute: {code_done-loop_start}")
302 r=input('Please press enter to continue.')
```