



University of
Zurich^{UZH}

New Approach to the Circle Hough Transform for the LHCb for Detecting Cherenkov Rings in the RICH detector

Master thesis
of
Philipp Gloor

Mathematisch-naturwissenschaftliche Fakultät
der
Universität Zürich

Prof. Dr. U. Straumann
Prof. Dr. P. Saha
Dr. O. Steinkamp

Zürich 2016

Abstract

This thesis explores possible algorithms for the detection of circles of Cherenkov photons which are produced by particles travelling through a RICH detectors in the LHCb experiment. These circles can be detected with the help of a Hough transform. There is a short introduction to the linear Hough transform (which detects straight lines) to give a rough idea about what the Hough transform can do. Then there is a brief look at the circle Hough transform (which detects circles). This part is split in 3 sections, discussing 1D, 2D and 3D Hough transforms. The 1D Hough transform can be used when the center is given and the radius needs to be found. The 2D Hough transform looks for the position of circle centers ((x, y) coordinate) for a given radius. The 3D Hough transform is used when neither center nor radius are known. This is the case for the data from LHCb where there are data points (photons from several circles and noise hits) and the algorithm has to find all the circles.

Apart from the standard Hough transform with a 3D accumulator space this thesis develops a new approach for a Hough transform. This approach works on the basis that each circle is defined by 3 points. Given 3 points one can calculate the circumcenter (so the center of the circle we are interested in) and the radius of the circumcircle.

Contents

Abstract	2
1 Introduction	7
1.1 LHC - Large Hadron Collider	7
1.2 LHCb	8
1.2.1 Particle identification	9
2 Theory	11
2.1 Cherenkov radiation	11
2.2 RICH detector	12
2.3 Hough transform	12
2.3.1 Linear Hough transform	12
2.3.2 Circle Hough transform	14
2.4 Datasets	15
2.5 Existing algorithm	15
3 Methods	17
3.1 Conventional Hough transforms	17
3.1.1 1D: Known center - find radius	17
3.1.2 2D: Known radius - find center	18
3.1.3 3D: All parameters unknown	22
3.2 Combinatorial triplet Hough transform	25
3.2.1 Generating the triplets	25
3.2.2 Calculating the Circle given 3 points	25
3.2.3 Finding the radius and center of a circle	26
3.2.4 Combinatorics	27
3.2.5 Possible optimisation: average radius of random circles in a unit square	28
4 Results	32
4.1 1D Hough transform results	32
4.2 2D Hough transform results	38
4.2.1 2D Hough transform, 2 circles, 0 background hits	40
4.2.2 2D Hough transform, 5 circles, 30 background hits	41
4.2.3 2D Hough transform, 6 circles, 200 background hits	41
4.3 3D Hough transform Results	45
4.3.1 3D: Overview of the results	46
4.4 Combinatorial triplet Hough transform results	46
4.4.1 Overview of the results	46
4.4.2 Removing duplicates	47
4.4.3 Different thresholds	49
4.4.4 Different cuts for circle selection	51
5 Conclusions	56

List of Tables

2.1	Comparison of photoelectron yields (N_{pe})	15
4.1	Radius scores for the different events. There is always a big difference between the highest score which determines the circle and the second highest score which is noise. In the example with 1 circle and 600 background the second highest score is relatively high which is due to the many background hits that coincidentally have the same center as the real circle but with a different radius than the real one	34
4.2	Radius scores for the different events. There is always a big difference between the highest score which determines the circle and the second highest score which is noise. In the example with 1 circle and 600 background the second highest score is relatively high which is due to the many background hits that coincidentally have the same center as the real circle but with a different radius than the real one. For the event with 6 circles and 200 backgrounds sometimes the second highest score is extremely close to the highscore. This happens mainly when two circles have a similar radius and thus the second highest score is not just noise but actually another circle.	38
4.3	Circle scores for 1 circle 600 background hits	46
4.4	Efficiencies for different thresholds	51
4.5	Reducing ghost rate for a fixed radius cut (0.003) with varying center cut. The tighter the center cut is the more duplicate circle remain after the . . .	52
4.6	Reducing ghost rate for a fixed center cut (0.003) with varying center cut. The tighter the center cut is the more duplicate circles fail being detected by the removeDuplicate code shown above	52
4.7	Results for all the cut combinations for radius and center cuts	53

List of Figures

1.1	The LHC ring with its 4 experiments: ATLAS, CMS, ALICE and LHCb . . .	7
1.2	LHCb Detector: The beams collide inside the Vertex Locator. The RICH1 is positioned before the tracking station (TT) and the magnet. RICH2 is set up after the magnet and before the silicon trackers (T1-T3) and the muon and calorimeter (M1-M5)	8
2.1	Cherenkov radiation where θ is equal to $\cos \theta = \frac{\frac{c}{n}t}{\beta ct}$	11
2.2	RICH-1 detector [8].	13
2.3	r - θ parametrisation	13
2.4	Example of a linear HT space	14
2.5	Example of a linear HT where points don't lie on a line	14
2.6	Number of points per circle (N_{pe}) for the 10'000 simulated events in the test data sample	16
2.7	Number of circles per event in the test data sample	16
2.8	The radius distribution of the test data events.	16
3.1	Normal Distribution: Used as a weight function	18
3.2	Pseudo Code 1D HT	18
3.3	Graph visualisation of a 1D radius histogram	19
3.4	Pseudo code 2D HT	20
3.5	2D weight matrix, first iteration	23
3.6	3D broadcasting	23
3.7	Calculation of weights and threshold check for 3D HT	23
3.8	Remove used points	24
3.9	The circumradius (R) and the circumcenter (P) of a circle defined by three points (A, B, C).	26
3.10	Complexity of the combinatorial triplet Hough transform	27
3.11	Number of combinations with $\binom{N}{3}$ compared to the number of combinations generated from $\binom{N/2}{3}$	28
3.12	The probability when splitting randomly a list of x points into two that one list has more than 10 points.	29
3.13	Radius distribution for background	31
4.1	These are the circles as generated by the simulation.	33
4.2	Circles found by the 1D Hough transform. The circle in Figure 4.2a has its center in the origin so the algorithm did find the circle	34
4.3	Radius scores for 1 circle and 600 background (top) and 2 circles with 0 background hits (bottom). For the event with only 1 circle and 600 backgrounds the scores from background hits for a given radius are higher than in the other events with less noise but the correct radius has still a distinct peak.	35
4.4	Radius scores for the 1D Hough transform for the 5 circles with 0 background.	36

4.5	5 circles, 30 background hits result	37
4.6	The first three events were solved correctly by 2D Hough transform	39
4.7	The rings generated by the simulation on the right and the rings found by the algorithm on the left. The magenta colored circle in the left image is incorrect. It replaced the yellow circle (taken from the simulated data on the left)	39
4.8	Center score for the 2D Hough transform for 2 circles with 0 background.	40
4.9	Two slices out of the 2D histogram	40
4.10	Center scores for all the centers for an event with 5 circles.	41
4.11	Center scores for 6 circles with 200 background hits.	42
4.12	On the left side the wrong result obtained by the 2D Hough transform and the correct one on the right side	42
4.13	2D HT: Comparison center scores 6 circles, 200 background hits	43
4.14	2D HT: Comparison result, 6 circles, 200 background hits	44
4.15	Circles found by the 3D Hough transform	45
4.16	Histogram of duplicates	48
4.17	Pseudo code for removing possible duplicates	49
4.18	The result of the combinatorial triplet Hough transform before cleaning up the duplicates. Blue and purple are basically the same circle.	50
4.19	Reversed cumulative distribution of the points per circle. The plot shows the ratio of circles with more than x points over the total number of circles	51

1 Introduction

1.1 LHC - Large Hadron Collider

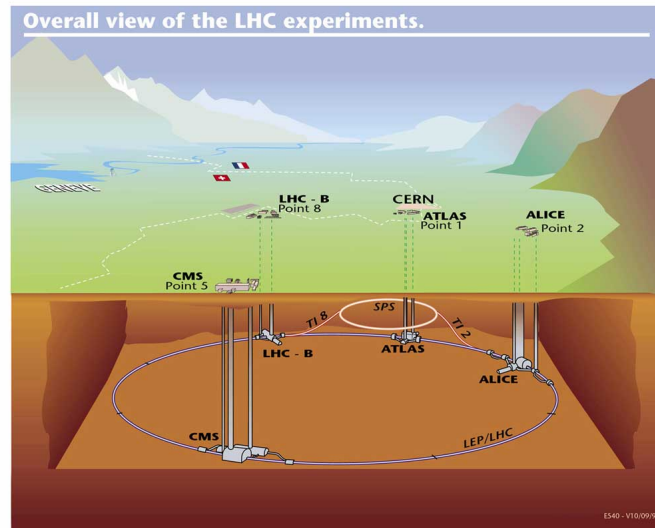


FIG. 1.1 – The LHC ring with its 4 experiments: ATLAS, CMS, ALICE and LHCb

The Large Hadron Collider (LHC) is the largest and highest-energy particle accelerator in the world colliding protons on protons at beam energies of up to 16.5 TeV and lead ions at beam energies up to 5.5 GeV/nucleon. It was built by the European Organisation for Nuclear Research from 1998 to 2008. It aims to test the predictions of different theories in high-energy particle physics, in particular for the search of the Higgs boson (which has been confirmed last year) and signs for new physics beyond the Standard Model of particle physics. The LHC lies in a tunnel 27 km in circumference and up to 100 m below the surface of the French-Swiss border near Geneva. The LHC was built in collaboration with over 10000 scientists and engineers from over 100 countries. The accelerator has been running with a center of mass energy $\sqrt{s} = 13$ TeV since 20 May 2015.

The LHC consists of 4 large experiments [1–4]:

ATLAS/CMS

- The two multi-purpose experiments at the LHC probing $p - p$ and heavy ions for direct searches of new particles.

ALICE

- ALICE (A Large Ion Collider Experiment) is a general-purpose, heavy ion detector at the CERN LHC which focuses on QCD, the strong interaction sector of the Standard Model e.g. for evidence for quark-gluon plasma.

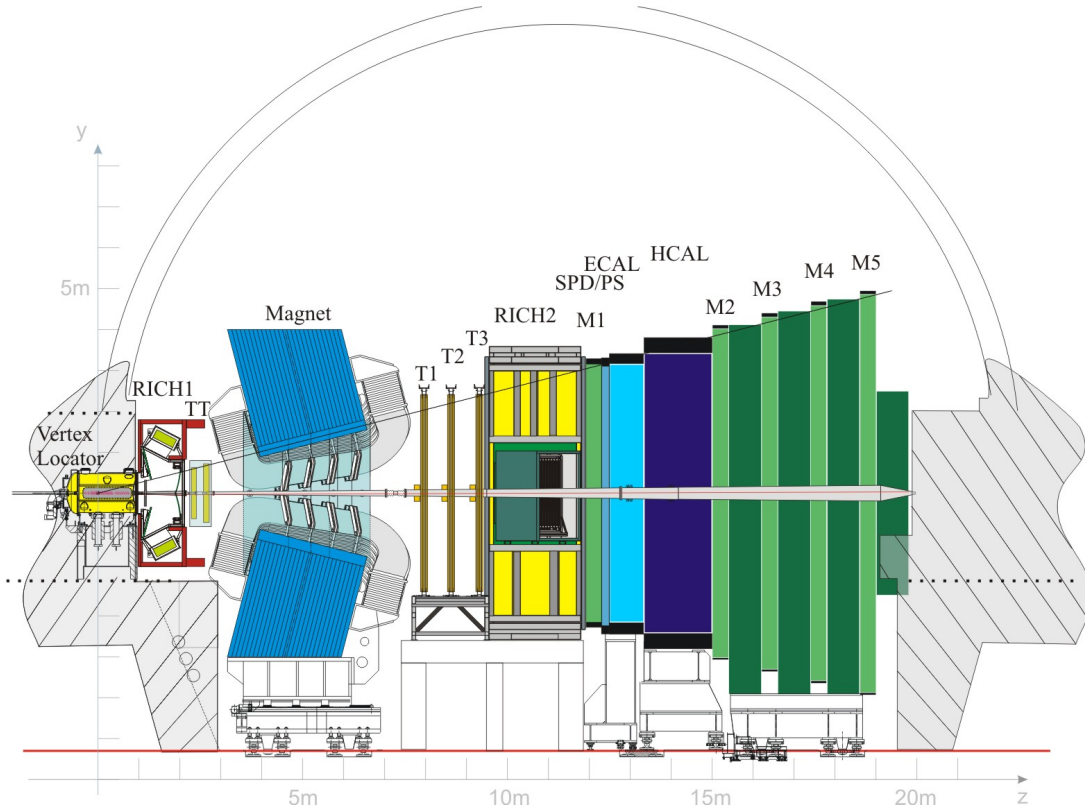


FIG. 1.2 – LHC b Detector: The beams collide inside the Vertex Locator. The RICH1 is positioned before the tracking station (TT) and the magnet. RICH2 is set up after the magnet and before the silicon trackers (T1-T3) and the muon and calorimeter (M1-M5)

LHC b

- LHC b is testing the Standard Model by confronting predictions with its precise measurements in CP violation and rare decays of particles containing b and c quarks.

1.2 LHC b

LHC b is one of the four big experiments conducted at the LHC (ATLAS, CMS and ALICE being the other 3). The main goal of this experiment is the study of decays of particles containing b and \bar{b} quarks (B-Mesons). During collisions these particles are produced mostly at small polar angles with respect to the beam axis. This is reflected in the design of the LHC b detector which is a forward arm spectrometer 20 meters long with subdetectors arranged along the beam pipe as shown in figure 1.2.

A quick overview of the detector parts [5].

VELO The VERtix LOcator surrounds the region where the beams collide and b/\bar{b} are produced. The VELO measures the distance between the $p - p$ collision point and the point where the B particles decay. B particles are short-lived (decaying after typically 1 cm thus the B particles are not measured directly but inferred from the separation of these two points and the properties of their decay products.

RICH The RICH detectors are built for particle identification in particular to distinguish charged kaons from pions. One detector on each side of the magnet is used to cover different momentum ranges. RICH detectors work by measuring emissions of Cherenkov radiation which is emitted if a particle travels faster than the speed of light through a certain medium (often compared to breaking the sound barrier). The emission angle depends on the speed of the particle, so knowing the speed and the momentum (from the curvature from the track induced by the magnet) the mass of the particle can be inferred.

Magnet A particle normally moves in a straight line but entering a magnetic field causes the path of charged particles to curve according to the Lorentz force

$$\mathbf{F} = q (\mathbf{E} + (\mathbf{v} \times \mathbf{B}))$$

thus allowing to determine the charge sign of the particle. Also the track curvature can be used to infer the momentum of the particle.

Tracking System The tracking system is based on 4 planar tracking stations. It is used to determine the momentum of charged particles by measuring the bending of the trajectory in the magnetic field. In the silicon detector a passing particle generates an electron-hole pair which deposits a charge on the silicon strips. In the gas-filled tubes a particle ionises the gas molecules which deposit charges on a wire.

Calorimeters They are designed to stop particles and measure their energy lost. The design of the stations is sandwich like. One metal plate and one scintillator plate. Interactions in the metal plate cause a secondary shower of charged particles which induce a scintillation light in the scintillator plate. The energy lost is proportional to the amount of light emitted. Calorimetry is also the main way of identifying particles with no charge (e.g. photons, neutrons).

Muon system Muons play an important role in many analyses. There are 5 planar stations at the end of the detector. The total area covered by these stations is about 435 m². The goal of the absorber is to stop all particles except muons who still can pass. The muons get detected in the gas chambers where their trajectory is measured.

1.2.1 Particle identification

An important requirement at LHCb is particle identification. This is handled by CALO, Muon and RICH sub-detectors. The Calorimeters beside measuring energies and positions of electrons, photons and hadrons also provide identification of said particles e.g. by measuring the shape of the induced showers. The Muon system identifies muons to a very high level of purity which is essential for many J/Ψ 's in their final states.

Hadron identification is very important for decays where the final states of interest are purely hadronic. The LHCb RICH system provides this, covering a momentum range of approximately 1–100 GeV. It is composed of two detectors. One positioned upstream of the dipole magnet and the other one positioned downstream of the dipole magnet. The optics is arranged similarly in both sub-detectors: spherical focusing mirrors project the Cherenkov photons onto a series of flag mirrors which then reflect them onto a series of

photon detector arrays, located outside the detector acceptance [6]. +Hadron identification, i.e. separation of kaons, pions and protons, is very important for many analyses. The LHCb RICH system provides this, covering a momentum range of approximately 1–100 GeV. It is composed of two detectors. One is positioned upstream of the dipole magnet and the other one positioned downstream of the dipole magnet. The optics is arranged similarly in both sub-detectors: spherical focusing mirrors project the Cherenkov photons onto a series of flat mirrors which then reflect them onto an array of photon detector, located outside the detector acceptance [6].

2 Theory

2.1 Cherenkov radiation

The speed of light in vacuum, c , is a universal physical constant. According to Einstein's special theory of relativity, c is the maximum speed at which all matter (or information) in the universe can travel. The speed at which light propagates in a medium, however, can be significantly less than c .

Cherenkov radiation results when a charged particle travels through a dielectric medium with a speed greater than the speed of light through said medium. The velocity that must be exceeded is the phase velocity (v_{Phase} or short v_p) and not the group velocity ($v_{\text{Group}} = \frac{\partial \omega}{\partial k}$).

$$v_p = \frac{\lambda}{T} \quad \text{or} \quad \frac{\omega}{k}$$

where λ is the wavelength of the light and T the period and ω the angular frequency and the wavenumber k .

As a charged particle travels through the medium, it disrupts the local electromagnetic field. If the particle travels slower than the speed of light then the disturbance elastically relaxes to the mechanical equilibrium as the particle passes. However, if the particle travels faster than the speed of light, the limited response speed of the medium means that a disturbance is left in the wake of the particle, and the energy in this disturbance radiates as a coherent shockwave.

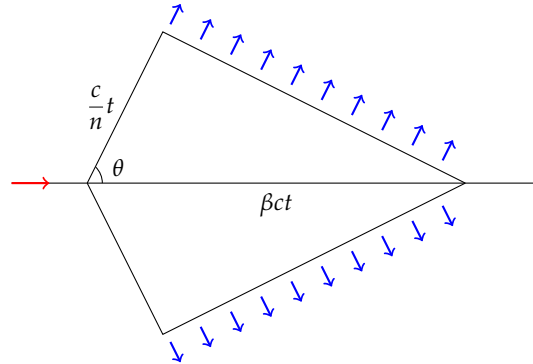


FIG. 2.1 – Cherenkov radiation where θ is equal to $\cos \theta = \frac{\frac{c}{n}t}{\beta ct}$

$$\begin{aligned} x_p &= v_p \cdot t = \beta ct \\ x_{\text{em}} &= v_{\text{em}} \cdot t = \frac{c}{n}t \\ \cos \theta &= \frac{x_p}{x_{\text{em}}} = \frac{\frac{c}{n}t}{\beta ct} = \frac{1}{n\beta} \end{aligned}$$

which is independent of time.

2.2 RICH detector

Particle identification is a fundamental requirement for many analyses at the LHCb experiment. The LHCb experiment is unique at the LHC in the sense that it uses RICH detectors for hadronic particle identification therefore much better than e.g. ATLAS or CMS. Using three different radiators the RICH detectors cover a wide range of momentum (1-100 GeV/c).

Both RICH-1 and RICH-2 are located in low magnetic field regions to keep the tracks straight while they pass through the radiators. They both also have a tilted spherical focusing primary mirror and a secondary flat mirror to limit the length of the detector along the beam pipe. The spherical focusing mirrors use the property that photons generated at a fixed angle with respect to the particle trajectory are then focused onto a ring in the photon detector plane (via the secondary flat mirror). The photon detectors are hybrid photon detectors (HPD) which have a resolution of $2.5 \text{ mm} \times 2.5 \text{ mm}$. The HPD are used to measure the spatial position of emitted Cherenkov photons. The HPD is a vacuum photon detector in which a photoelectron, released from the conversion in a photocathode of an incident photon, is accelerated by and applied voltage onto a silicon detector[3]. The total area of the two detector planes in the RICH-1 detector is $1302 \text{ mm} \times 555 \text{ mm}$ each and $710 \text{ mm} \times 1477 \text{ mm}$ for RICH-2.

The RICH-1 in front of the magnet covers a lower momentum range from 1-60 GeV/c. It is composed of 5 cm thick aerogel tiles arranged around the beam pipe. The aerogel with $n = 1.03$ is suited for the lowest momentum tracks. Directly behind the aerogel is circa 1 m of C_4F_{10} ($n = 1.0014$) which covers the intermediate region of momentum. For the highest momentum tracks, gaseous CF_4 ($n = 1.0005$) is used in the RICH-2 [7].

There is a strong correlation between the polar angle and momentum of the particles. Particles with a larger polar angle tend to have lower momentum. That is why RICH-1 with the aerogel is located before the dipole magnet so tracks with low momentum will be covered before they swept out of the acceptance by the magnet.

2.3 Hough transform

The Hough transform [9] is a feature extraction technique used in image analysis, computer vision and digital image processing.

The purpose is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space from which object candidates are obtained as local maxima in a so called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

Initially the Hough transform was concerned with finding straight lines [10] but has been extended to identifying positions of more complicated shapes, such as circles and ellipses.

2.3.1 Linear Hough transform

A linear function is often defined as:

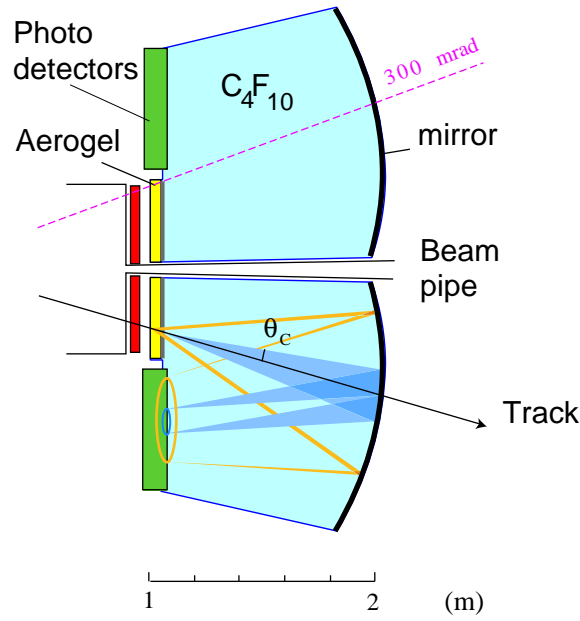


FIG. 2.2 – RICH-1 detector [8].

$$f(x) = m \cdot x + b$$

where m is the slope of the line and b the intercept. For the Hough transform however, this representation is not ideal. For a vertical line m would go to infinity which gives us an unbound transform space for m . For this reason Duda and Hart suggested the r - θ parametrization [9].

$$r = x \cos \theta + y \sin \theta \quad (2.1)$$

where r is the distance from the origin to the closest point on the line and θ is the angle between the x -axis and the line connecting the origin with that closest point.

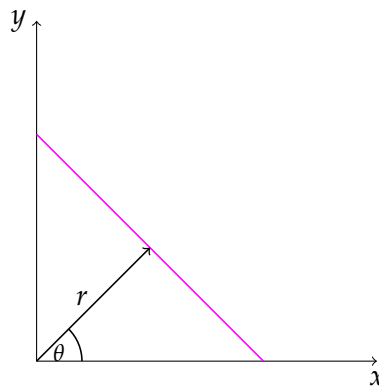


FIG. 2.3 – r - θ parametrisation

This means given a single point in the (x, y) plane, the set of all lines going through this point form a sinusoidal curve in r - θ space. Another point that lies on the same straight line in the (x, y) plane will produce a sinusoidal curve that intersects with the other at (r, θ) and so do all the points lying on the same straight line.

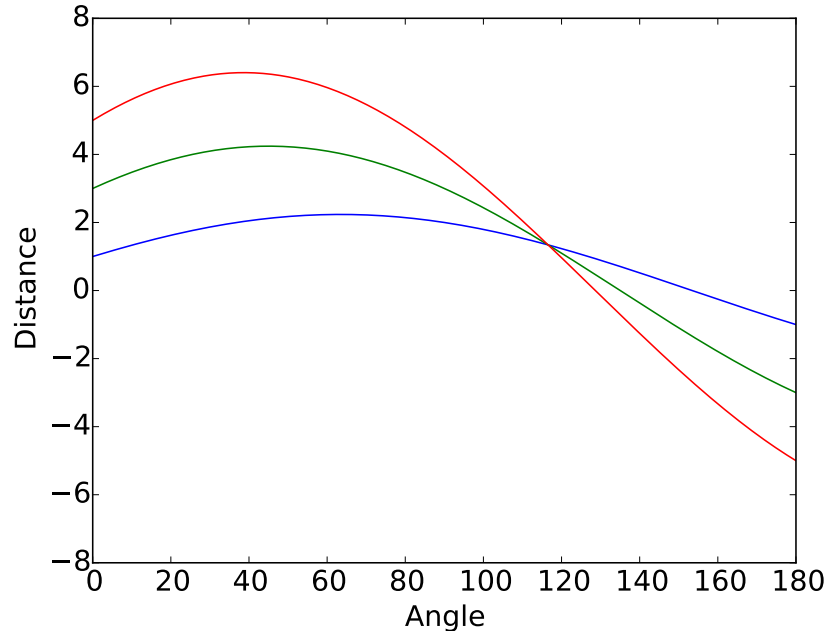


FIG. 2.4 – Example of a linear HT parameter space. Each line is representing a point of the line. For each point several lines are drawn for different angles according to equation 2.1 so this plot draws the perpendicular distances from these lines to the origin. When different lines intersect it means that these 3 Points lie on a line with the parameters given by the plot.

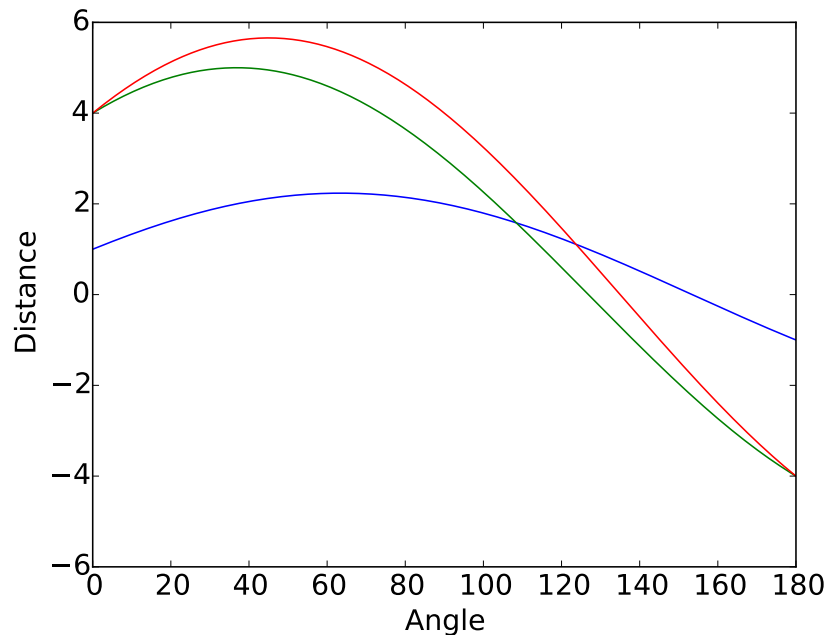


FIG. 2.5 – Example of a linear HT where points don't lie on a line. Since any two points can form a line there are still intersections but never more than 2.

2.3.2 Circle Hough transform

For this thesis we are interested in circle detection so we need to adapt the linear Hough transform in order to find circles. In a two dimensional space (x, y) , a circle can be de-

Radiator	N _{pe} from data		N _{pe} from simulation	
	tagged D ⁰ → K ⁻ π ⁺	pp → pp μ ⁺ μ ⁻	Calculated N _{pe}	true N _{pe}
Aerogel	5.0 ± 3.0	4.3 ± 0.9	8.0 ± 0.6	6.8 ± 0.3
C ₄ F ₁₀	20.4 ± 0.1	24.5 ± 0.3	28.3 ± 0.6	29.5 ± 0.5
CF ₄	15.8 ± 0.1	17.6 ± 0.2	22.7 ± 0.6	23.3 ± 0.5

TAB. 2.1 – Comparison of photoelectron yields (N_{pe}) determined from D* → D⁰ π⁺ decays in simulation and data, and pp → pp μ⁺ μ⁻ events in data, using the selections and methods described in the text [11]

scribed by:

$$(x - c_x)^2 + (y - c_y)^2 = r^2 \quad (2.2)$$

Where (c_x, c_y) is the center of the circle and r the radius. The possible parameters for the parameters space are now c_x, c_y and r . This means if we know the center of the circle the parameter space is one-dimensional and if we know the radius of the circle the parameter space is two-dimensional and of course if we know nothing the parameter space is three-dimensional.

2.4 Datasets

Two Monte-Carlo data sets were used for this thesis. One was a training data set of 250 simulated events and the test data on which the performance of the algorithm was tested, consisting of 10'000 simulated events with a total of 49'979 circles across all events. The properties of these events (radius, number of points per circle, number of circles per event) are taken from both simulation of LHCb events and real data taken from the experiment.

The number of points per circle is determined by the photoelectron yield N_{pe}. It is measured for both *normal* events and *ideal* events. The normal event is representative of nominal RICH running conditions during LHCb physics detector operations. The ideal event is a special event type with very low background hits [11].

Also to get an idea what the average size of a Cherenkov ring is can be seen in [12]. Rings in RICH-1 have radiuses generally smaller than 0.15 m (given they are from the C₄F₁₀ radiator; rings from the aerogel have slightly bigger radiuses – around 0.20 m). All generated rings in the test data have a radius that is smaller than 0.15 m.

2.5 Existing algorithm

There is already an algorithm in place in LHCb [12]. The disadvantage of the existing algorithm is due to the fact that it uses tracks to reconstruct the circles. All charged tracks that are reconstructed by the tracking system are reflected at the RICH mirrors in order to give a center for a Cherenkov ring. In order to match tracks with rings each ring is matched with the track that has its position closest to the center of the ring. This means that this method uses information from outside the RICH detector to construct the circles.

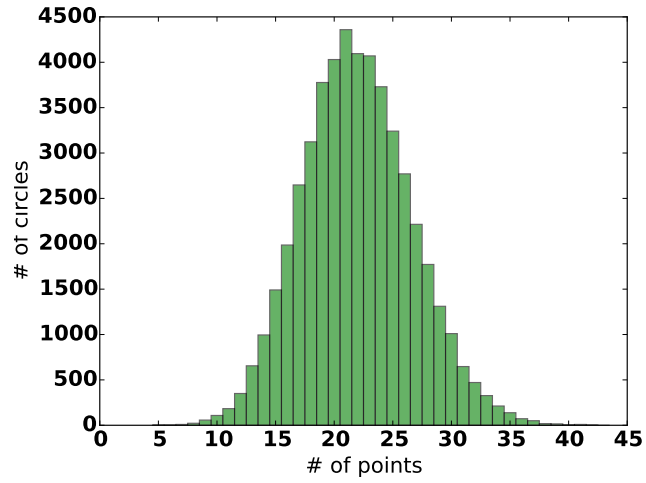


FIG. 2.6 – Number of points per circle (N_{pe}) for the 10'000 simulated events in the test data sample

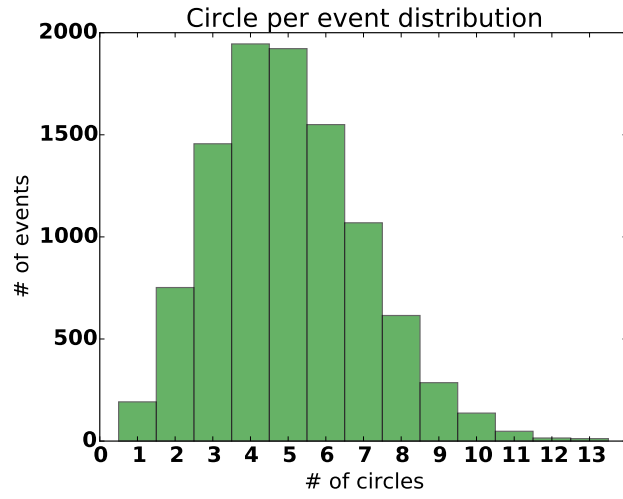


FIG. 2.7 – Number of circles per event in the test data sample

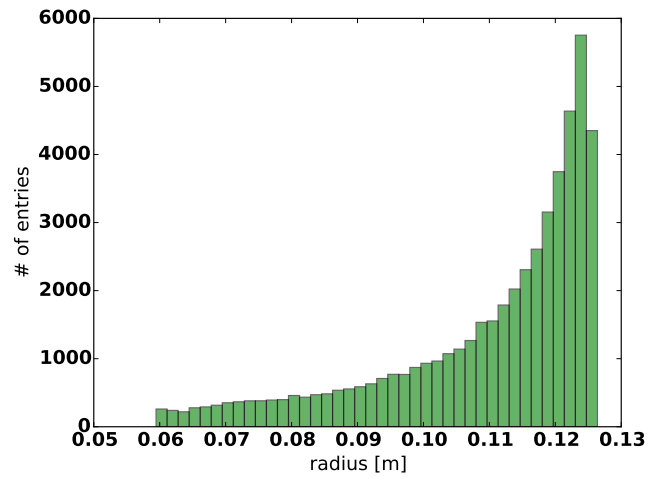


FIG. 2.8 – The radius distribution of the test data events.

3 Methods

3.1 Conventional Hough transforms

In the following subsections we discuss the conventional Hough transforms for the case of a one, two and three dimensional parameter space. These methods were mainly considered to get an idea of what was possible with the conventional Hough transform. For closer study the method of choice was the combinatorial triplet Hough transform discussed in depth in section 3.2.

3.1.1 1D: Known center - find radius

In this case the center(s) of the circle(s) is/are known so only the radius is missing. For the radius an array is defined with a minimum value and increasing stepsize to the maximum possible radius. For the example studied here the minimum was chosen to be 0, the maximum radius to be 1 and the stepsize equal to 0.001. The following scoring function $\eta(r)$ was used to calculate the distance of a data point (x, y) from the given center (c_x, c_y) and radius r :

$$\eta(r) = (c_x - x)^2 + (c_y - y)^2 - r^2 \quad (3.1)$$

If $\eta(r) = 0$ means that the particular point (x, y) would lie on the circle of radius r with center (c_x, c_y) . A Gaussian distribution 3.2 is used to have a well defined value for such point. For a bigger σ a point (x, y) that is not exactly on the circle will still contribute to the total score. The smaller the σ the tighter the range is in which a point will be considered lying on a circle or not.

$$w(\eta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\eta^2}{2\sigma^2}\right) \quad (3.2)$$

The value from equation 3.2 is added to the bin in the radius histogram which corresponds to the r used to calculate the weight.

Equation 3.2 is of course just the known coordinates of the circle equation with c_x, c_y being the center of the circle, x, y are the data points and r is the radius. If a lot of the data points have the same distance r from the circle center there will be a high score for this particular radius. The index for the highest score can then be used to find the corresponding radius. The pseudo code for this Hough transform is shown in figure 3.2. An example of a resulting radius histogram is shown in figure 3.3

Complexity

The complexity of this algorithm is of $\mathcal{O}(n)$ where n is the number of bins in the radius array.

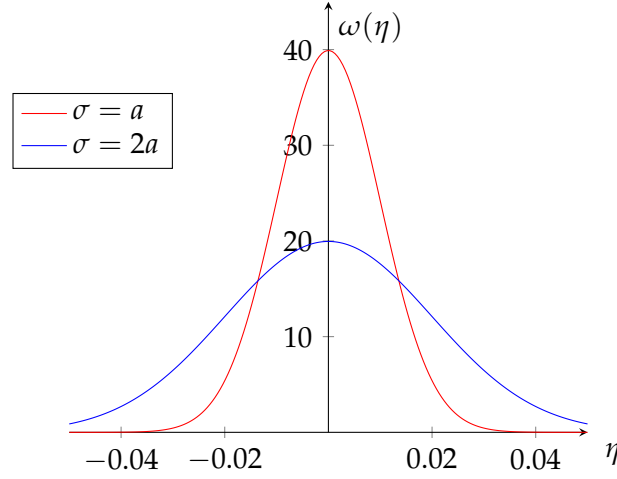


FIG. 3.1 – Using the probability density function of the normal distribution to calculate the score of a point in order to have a well defined maximum if a point lies directly on the circle and $\eta(r) = 0$. Increasing the σ increases the width of the function and by extension increases the score of points who are not lying directly on the circle

```

DIMENSION = 1001
r= linspace(0,1,DIMENSION)
for c in centers:
    scores = zeros(DIMENSION)
    for x,y in allPoints:
        s = 2*BIN_WIDTH
        eta = (c_x-x)**2 + (c_y-y)**2 - r**2
        scores += 1. / ( sqrt( 2 * Pi ) * s )
                    * exp( -( eta ** 2 )
                      / ( 2 * s ** 2 ) )
    index = max(scores)
    circle = {}
    circle['center'] = c
    circle['radius'] = r[index]

```

FIG. 3.2 – Pseudo code for the 1D Hough transform. r is an array of length 1001 so η will also be an array of length 1001. Scores is where the score for each iteration is stored. For each point the score is computed and added to the scores array and at the end the index with the highest score is the index we need to get the radius

3.1.2 2D: Known radius - find center

In this case the radius is known and the x and y coordinates of the center (c_x, c_y) are unknown. Now, the accumulator is 2 dimensional. The range of that space is the dimension of the detection plane which for this thesis is $[-0.5, 0.5]$. The size of the bins is chosen to be 0.001 since the known radius given from the test data was of the same order. So if the detection plane was in reality 1 m by 1 m the binning of the accumulator in each dimension is 1 mm. As in the one dimensional case we use the scoring function 3.1 in combination with the weight function 3.2.

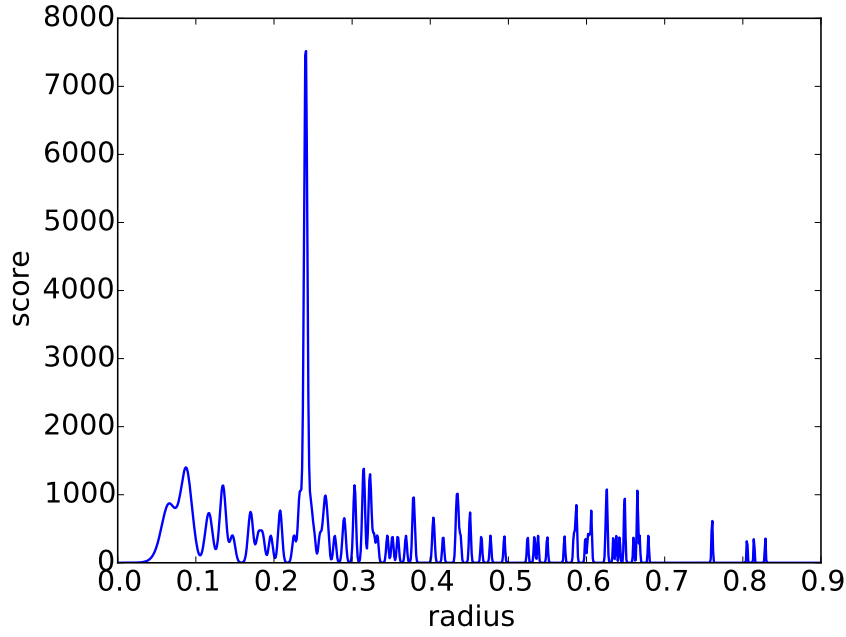


FIG. 3.3 – A graph visualisation of a 1D radius histogram. The peak between 0.2 and 0.3 has by far the highest score and the bin position of the peak is the radius candidate for a given circle center (c_x, c_y)

Complexity

The complexity of this algorithm is $\mathcal{O}(n \times m)$ where n and m are the number of bins for one coordinate respectively of the histogram. The calculation of the weight has to be done for each data point of the 2D histogram. So in a 1000×1000 histogram with 400 data points we calculate 400'000'000 times the weight of a grid point. Reducing the dimensions of the histogram weakens the accuracy of the whole algorithm but can speed up the calculations considerably. With a 1000×1000 histogram the resolution in each space dimension is 1 mm. The RICH Technical Design Report states the resolution of the HPD is $2.5 \text{ mm} \times 2.5 \text{ mm}$.

The need (not entirely true – see below) to calculate the weight for each grid point and data point means that there is a loop over data points and two loops for the x and y coordinate of the grid. To improve upon that there is the possibility of array broadcasting.

```

DIMENSION = 1001
xbins = linspace(-0.5,0.5,DIMENSION)
ybins = linspace(-0.5,0.5,DIMENSION)
x, y = broadcast_arrays( xbins[..., newaxis],
                        ybins[newaxis,...] )

for r in Radiuses:
    weights = zeros( (DIMENSION,DIMENSION) )
    for xd,yd in allPoints:
        s = 2*BIN_WIDTH
        eta = (xd-x)**2 + (yd-y)**2 - r**2
        weights += 1. / ( sqrt( 2 * pi ) * s )
                    * exp( -( eta ** 2 )
                      / ( 2 * s ** 2 ) )
    i, j = argmax(weights)
    removeUsedPoints()
    circle['Center'] = (xbins[i], ybins[j])
    circle['Radius'] = r

```

FIG. 3.4 – Pseudo code for the 2D Hough transform. `xbins` and `ybins` are arrays of length 1001. Here we use array broadcasting in order to avoid for loops and the weights can be evaluated in one line. This means that the `x` and `y` variables have dimension (1001,1001) but they don't take up that much memory. The `x` variable for example just broadcasts its value from the first row down to all the other rows and for `y` it broadcasts the first column to all the other columns. The variable `weights` is a 1001 by 1001 matrix. Again the entry with the highest score is the candidate for a possible circle center and if found is stored in a final variable called `circle`.

Array broadcasting

Consider following one dimensional arrays where x is a 1D histogram binning entries from 1 to 4 and same for y .

$$x = [1, 2, 3, 4]$$

$$y = [1, 2, 3, 4]$$

Now all combinations between an element of x and y represent a 2D grid $((1, 1), (1, 2), \dots)$. So to iterate through all those grid points one would have to create 2 for-loops iterating through x and y as shown in below pseudo code snippet:

```

def noBroadcast():
    a = np.random.randn(100)
    b = np.random.randn(100)
    for x in a:
        for y in b:
            print (1-x)^2 + (2-y)^2 - 9

```

This is not only slow but also doesn't look too nice if there are even more loops. Broad-

casting now turns the one dimensional arrays of length n into two n by n matrices

$$x = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

and

$$y = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

And with this the loops can be omitted:

```
def withBroadcast():  
    a = np.random.randn(100)  
    b = np.random.randn(100)  
    x,y = np.broadcast_arrays(a[... , np.newaxis],  
                              b[np.newaxis, ...])  
    print (1-x)^2 + (2-y)^2 - 9
```

In this case this prints a 4 by 4 array with the function evaluated for each combination of entries of x and y

$$\begin{bmatrix} -8 & -9 & -8 & -5 \\ -7 & -8 & -7 & -4 \\ -4 & -5 & -4 & -1 \\ 1 & 0 & 1 & 4 \end{bmatrix}$$

A runtime comparison shows

```
In [3]: %timeit withBroadcast()  
10000 loops, best of 3: 76.8 us per loop
```

```
In [4]: %timeit noBroadcast()  
100 loops, best of 3: 7.99 ms per loop
```

So the version with broadcasting is 100 times faster than the double loop. And the memory consumption is moderate since the broadcasted entries aren't new memory locations but just refer to the initial array.

Optimizations

It was mentioned before that for each data point the weight for the whole grid has to be calculated. This is not true. In the 2D case each grid point is a potential center for a circle only if it is not further away and a threshold radius R_T , so if a grid point is further away than this threshold radius this calculation could be skipped. This could probably be done even smarter with the use of a sub grid so only points in the surrounding sub grids are considered of being possible centers and not the whole space.

Simple example: 2 circles without background

3.1.3 3D: All parameters unknown

In this case all that is known are the data points and the algorithm has to retrieve both the center and the radius of the circles. The accumulator space is now in three dimensions, two for the coordinates of the center and one for the radius. Similar to the 2D case, array broadcasting (see code snippet 3.6) is used to speed up the calculations of the weights. Furthermore, the algorithm has to decide itself whether or not all circles have been found since there unlike in the previous two cases, there isn't any information available about the circles so a condition has to be set to decide when there are no more circles.

For deciding whether or not the algorithm has found all rings, a simple score threshold is used that whenever the highest score of the weight matrix is less than the threshold the algorithm stops and it is assumed that all circles have been found. The pseudo code for calculating the weights and the threshold check is shown in 3.7. In the 3D case the algorithm has to remove data points that have contributed to the highest score. If not done the algorithm will loop endlessly because always the same score will be found. Once the highest score has been found the center coordinates and radius are extracted from the histogram and with these the algorithm calculates for every data point, if this data point lies on that circle given the extracted center and radius. If the data point lies within two times the bin width of the circle, it will be removed (see 3.8).

As before a scoring function is used but this time the scoring function is of the form $\eta(x, y, r)$ and each point in weights then stands for the score of the x, y, r entries and their respective value.

$$\eta(x, y, r) = (x - d_x)^2 + (y - d_y)^2 - r^2 \quad (3.3)$$

where d_x and d_y are the data points.

$$w(\eta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\eta^2}{2\sigma^2}\right) \quad (3.4)$$

Complexity

The complexity of this algorithm is of order $\mathcal{O}(m \times n \times r)$. As in the case the 1D and 2D Hough transform the accuracy directly depends on the binning. For the 3D Hough transform to match the resolution of the HPD of the LHCb RICH detector a binning of 400 makes sense if the detector dimensions are $1 \text{ m} \times 1 \text{ m}$.

Optimisations

As for the 2D HT introducing a sub grid can be introduced for the x, y plane so only grid points in the vicinity of a data point are used for calculating the score.

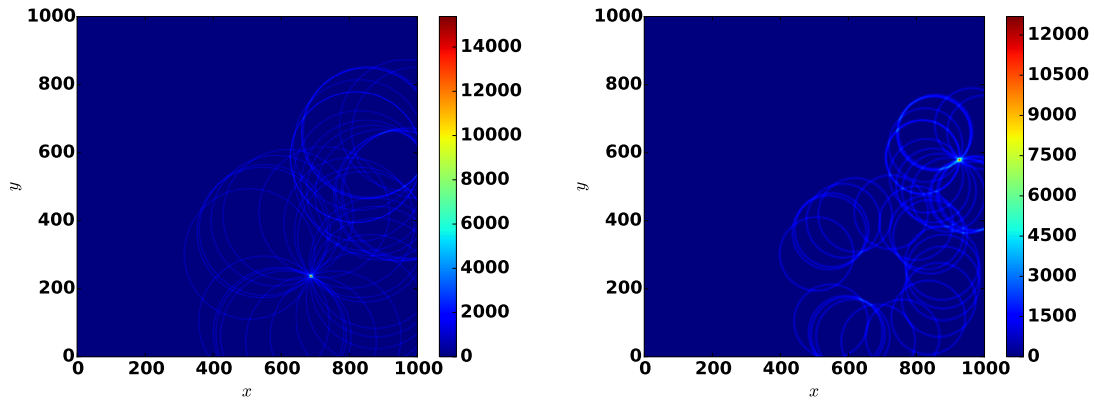


FIG. 3.5 – (Left) 2D weight matrix in the first iteration of the Hough transform algorithm. (Right) Second iteration of the 2D weight matrix of the Hough transform. Points that satisfied the condition being less than a certain ϵ away from the radius found in the first iteration are removed leaving (hopefully) only points available that belong to the second circles

```

xbins = np.linspace(-0.5,0.5,DIMENSION)
ybins = np.linspace(-0.5,0.5,DIMENSION)
rbins = np.linspace(0,0.5, R_DIMENSION)

x,y,r = np.broadcast_arrays(\
    xbins[np.newaxis,...,np.newaxis],\
    ybins[np.newaxis,np.newaxis,...],\
    rbins[...,np.newaxis,np.newaxis])

```

FIG. 3.6 – Broadcasting of the 3 arrays x, y, r . With this 3 for-loops can be avoided improving speed and readability of the code.

```

while True:
    weights = np.zeros(\
        (R_DIMENSION, DIMENSION, DIMENSION))

    for x0,y0 in data['allPoints']:
        s = 0.001
        eta = (x-x0)**2 + (y-y0)**2 - r**2
        weights += 1./( sqrt( 2 * sconst.pi ) * s )*\
            np.exp( -( eta ** 2 ) / \
                ( 2 * s ** 2 ) )
        index = np.argmax( weights )
        rr,ii,jj = np.unravel_index( index,
            (R_DIMENSION, DIMENSION, DIMENSION))
        score = weights[rr][ii][jj]
        if score < THRESHOLD:
            break

```

FIG. 3.7 – The while loop works as long as the found score is higher than THRESHOLD. If the score is lower than the threshold the loop breaks and the function returns the results that have been found.

```

circle['center'] = (xbins[ii], ybins[jj])
circle['radius'] = rbins[rr]
circles.append(circle)

used_xy += [tup for tup in data['allPoints'] if
    abs( ( tup[0] - circle['center'][0] ) ** 2 +
        ( tup[1] - circle['center'][1] ) ** 2 -
        circle['radius'] ** 2 ) < 2 * 0.001]
data['allPoints'][:] =
    [tup for tup in data['allPoints'] if
    abs( ( tup[0] - circle['center'][0] ) ** 2 +
        ( tup[1] - circle['center'][1] ) ** 2 -
        circle['radius'] ** 2 ) >= 2 * 0.001]

```

FIG. 3.8 – In order to avoid finding the same circle over and over again the algorithm has to remove points that belong to a ring. If a data point lies within two times the bin width of a circle the algorithm considers that data point to be part of that circle and removes that data point from the list.

3.2 Combinatorial triplet Hough transform

A circle is uniquely defined by 3 points and its radius and center can be calculated from the coordinates of these points. If there are 15 points lying on the same circle there are 455 possible combinations of triplets According to the binomial distribution.

$$\binom{N}{3} = \frac{N!}{k!(N-k)!}$$

Calculating the center and radius for these 455 triplets should result in the same center (x, y) and same radius r for all the triplets (floating point inaccuracy not considered).

Having one background hit in addition to the 15 circle hits increases the number of triplets number to 560. The triplets with points solely consisting of points on the circle still have the same center and radius but the new combinations that now include a background hit will vary and it is unlikely that any two triplets that include the background point will have the same center and radius. Here is an overview of the algorithm studied in this thesis.

1. Build all possible triples of points given the data points
2. For all triplets calculate the center and the radius of the potential circle
3. Remove all triplets that yield a radius bigger than a certain threshold.
4. Create a histogram with the distribution of r for the remaining triplets. Peaks in this histogram hint to a circle.
5. Scan the radius histogram for peaks and look at the 2D histogram of (x, y) for the triplets belonging to a given peak in the r distribution. If there is also a peak in the (x, y) histogram the set of the points of the triplets lie on a circle with a radius and center given by the peaks in the r and (x, y) histograms.

3.2.1 Generating the triplets

To generate the triplets the built-in function `itertools.combinations()` of python is used. The input is an iterable, in our case a list of tuples (each tuple is the x and y coordinate of a data point) which is used to create all possible combinations of triples (of said tuples).

3.2.2 Calculating the Circle given 3 points

Let (A, B, C) be a triplet of points in a 2D plane and a, b, c the lengths of the sides opposite to the respective corner. The semiperimeter is defined as

$$s = \frac{a + b + c}{2}. \quad (3.5)$$

Using this we can calculate the radius R of the circumcircle of the triangle \overline{ABC} :

$$R = \frac{abc}{4\sqrt{s(a+b-s)(a+c-s)(b+c-s)}} \quad (3.6)$$

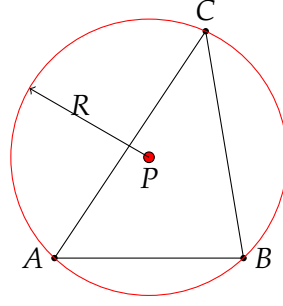


FIG. 3.9 – The circumradius (R) and the circumcenter (P) of a circle defined by three points (A, B, C).

We have $\lambda_1, \lambda_2, \lambda_3$ as the barycentric coordinates of the circumcenter:

$$\lambda_1 = a^2 \cdot (b^2 + c^2 - a^2) \quad (3.7)$$

$$\lambda_2 = b^2 \cdot (a^2 + c^2 - b^2) \quad (3.8)$$

$$\lambda_3 = c^2 \cdot (a^2 + b^2 - c^2) \quad (3.9)$$

Multiplying a matrix consisting of the column vectors of A, B, C with a column vector of $\lambda_1, \lambda_2, \lambda_3$ and dividing the resulting vector by the sum of the barycentric coordinates (for normalization) yields the coordinate of the circumcenter of the triangle \overline{ABC}

$$\begin{pmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \mathbf{P}' \quad (3.10)$$

$$\frac{\mathbf{P}'}{\lambda_1 + \lambda_2 + \lambda_3} = \mathbf{P} \quad (3.11)$$

3.2.3 Finding the radius and center of a circle

Once R and \mathbf{P} are known these values are stored as a pair in a tuple. A list holds then all tuples of (R, \mathbf{P}) values for all the triplets.

The next step is binning this data for R . This is done in a way that the center data of a tuple is associated with the R bin of the tuple. This allows the algorithm when looking for a peak in the R histogram to access the center data for this R value so essentially it splits the problem first in a 1D Hough transform where the algorithm looks for a peak in the radius distribution and with a given radius the algorithm can then search the (x, y) space for a peak and thus determine the center.

Finding the radius

Once the radius histogram is created the algorithm looks for the index where the maximum value of the histogram is stored. The algorithm checks first if the sum of the entry at the index and the left and right neighbour exceeds the radius threshold which is a tuning parameter. If the value exceeds the threshold the algorithm extracts the center data from the index and its left and right neighbour respectively.

Finding the center

Having a list of (x, y) points and one radius the algorithm now looks for a peak in the (x, y) plane. If the maximum found exceeds the center threshold then radius and the center coordinates are saved as a circle candidate.

3.2.4 Combinatorics

The main drawback of this method is that the combinatorics increase with a high number N of data points as $\binom{N}{3}$. For example, for 200 data points the number of triplets is

$$\binom{200}{3} = 1'313'400$$

and for 300 data points it is

$$\binom{300}{3} = 4'455'100.$$

The execution time of the algorithm is roughly of the order of $\mathcal{O}(N^3)$ which can be easily seen when taking the upper bound of $\binom{N}{3} \leq \frac{N^3}{3!}$ (see figure 3.10).

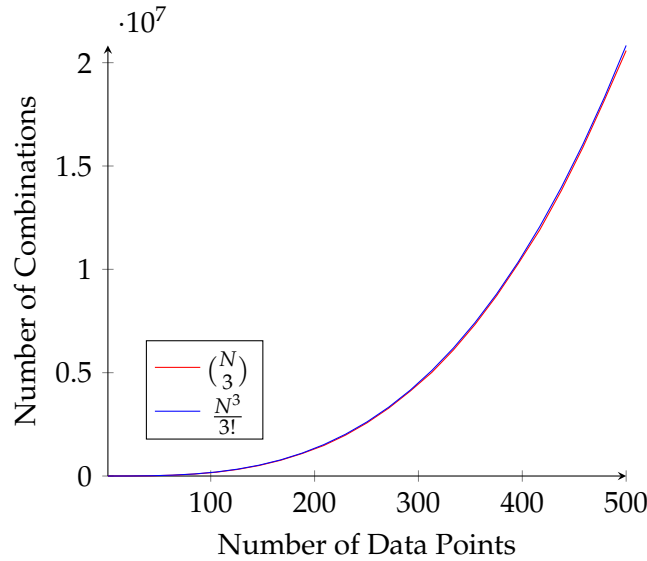


FIG. 3.10 – Scaling of the algorithm. Binomial Growth with $\binom{N}{3}$ compared with the approximation $\frac{N^3}{3!}$ where N is the number of data points. So the algorithm runs in the order of $\mathcal{O}(N^3)$

Optimisation

The algorithm needs a threshold on the minimum height of the peak in the (x, y) and r histograms in order to decide if a candidate is a circle or not. The threshold is defined as such that the minimum number of points on a circle has to be $\binom{T}{3}$ in order to be considered as a circle candidate.

But not only the number of triplets generated is a speed bump for the execution time but also the time it takes to create the triplets scales with $\frac{N^3}{3!}$. If there was a way to reduce not only the number of triplets created but also the time needed to create them would speed up the algorithm considerably.

This leads to the following idea: to split the original data set randomly into two lists. For each of these lists all possible combinations of triplets is generated separately and the two lists are then merged in one single list. In Figure 3.11 the difference in combinatorics between $\binom{N}{3}$ and $2\binom{N/2}{3}$ can be seen.

The problem with this approach is the possible loss of information and efficiency, since the algorithm has a threshold that defines how many entries a bin in the (x, y) and r histograms must have in order to be accepted as a candidate circle. This threshold should be high enough that triplets that contain noise points do not contribute significantly to the circle candidates but low enough that real circles with a low number of points can still be found.

So the question is what is probability of splitting points that belong to a specific circle in such way that at least one list has enough points that their combinations can reach the threshold e.g. if the threshold is 120 hits in the histograms this corresponds to 10 points ($\binom{10}{3} = 120$) having at least the number of points needed so the number of their triplets will be enough to pass the threshold. As soon as a circle has 20 points this becomes moot as always one list will have more than 10 points (see Figure 3.12).

In Figure 3.12 it can be seen that even if we split the lists there is a more than 50% chance that we lose no information once a circle has 13 points.

3.2.5 Possible optimisation: average radius of random circles in a unit square

An interesting property of calculating the radius of triplets generated from points that are distributed uniformly in the unit square is that they always obey a certain shape.

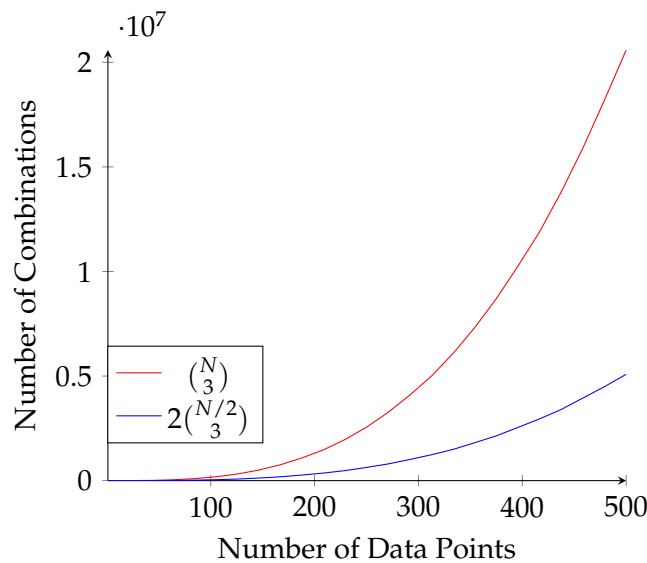


FIG. 3.11 – Number of combinations with $\binom{N}{3}$ compared to the number of combinations generated from $2\binom{N/2}{3}$

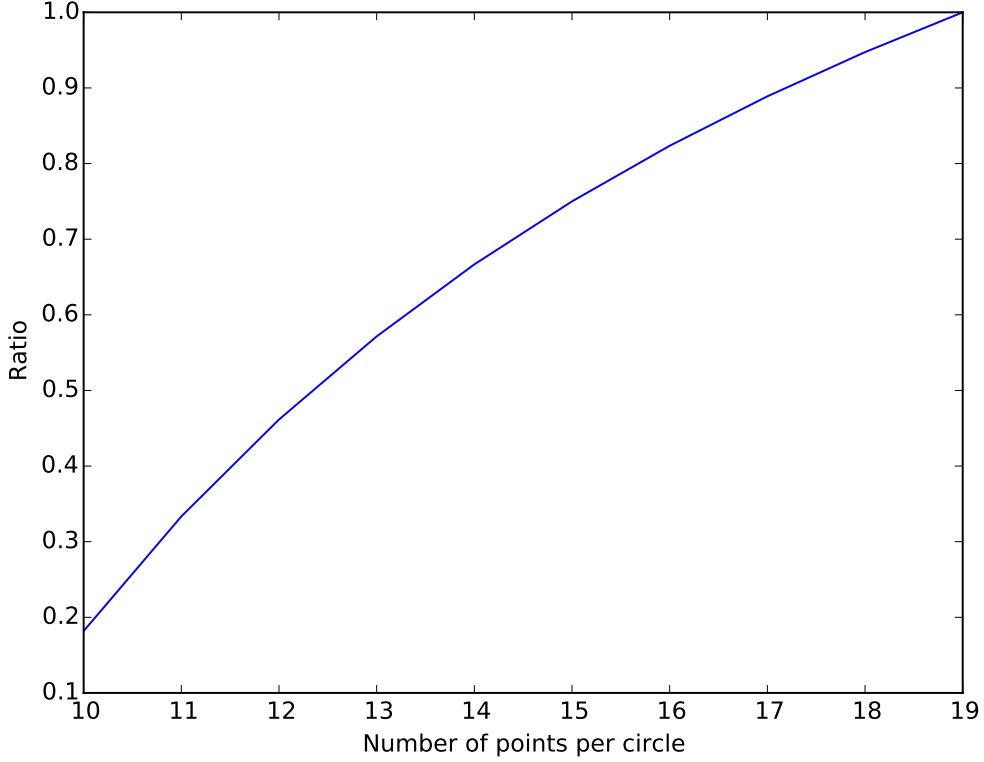


FIG. 3.12 – The probability when splitting randomly a list of x points into two that one list has more than 10 points.

To prove this the expected area of a triangle formed by three points randomly chosen from the unit square¹ has to be calculated. Let $A = (x_A, y_A)$, $B = (x_B, y_B)$, $C = (x_C, y_C)$ be the vertices of the random triangle T . We consider the case where $y_A > y_B > y_C$ which takes $\frac{1}{6}$ of the total “Volume”. Fix y_A, y_B, y_C for the moment and we can write.

$$y_B = (1 - t)y_A + ty_C, \quad 0 \leq t \leq 1.$$

The side AC of T intersects the horizontal level $y = y_B$ at the point $S = (s, y_B)$ with

$$s = s(x_A, x_C, t) = (1 - t)x_A + tx_C \quad (3.12)$$

The area X of T is then given by

$$X = \frac{1}{2}|x_B - s|(y_C - y_A)$$

We now start integrating with respect to our six variables. The innermost integral is with respect to x_B and gives

$$\begin{aligned} X_1 &:= \int_0^1 X dx_B = \frac{1}{2}(y_C - y_A) \left(\int_0^s (s - x_B) dx_B + \int_s^1 (x_B - s) dx_B \right) \\ &= \frac{1}{4}(y_C - y_A)(1 + 2s + 2s^2) \end{aligned}$$

¹This proof is taken from [13]

Next we integrate over y_B :

$$X_2 := \frac{1}{4} \int_0^1 \int_{y_A}^1 (y_C - y_A)^2 dy_C dy_A \times \int_0^1 \int_0^1 \int_0^1 (1 - 2s + 2s^2) dt dx_C dx_A$$

This gives

$$X_3 = \frac{1}{4} \cdot \frac{1}{12} \cdot \frac{11}{18} = \frac{11}{6 \cdot 144}$$

But generalizing our assumption at the beginning $y_A < y_B < y_C$ we multiply this result by 6 and obtain then $\frac{11}{144}$.

From [14] follows that the average area of a triangle in a unit circle is $\frac{3}{2\pi}$ so dividing this area by the area of the unit circle gives the expected area covered by a triangle in an arbitrary circle

$$\frac{\text{Expected area of a triangle in a unit circle}}{\text{Area of the unit circle}} = \frac{\frac{3}{2\pi}}{\underbrace{\pi r^2}_{r=1}} = \frac{3}{2\pi^2}$$

So finally the average area of a random circle can be obtained by dividing the average area of a triangle from the unit square by expected area covered by a triangle in an arbitrary circle so essentially the average area of a random circle within the unit square:

$$\frac{\frac{11}{144}}{\frac{3}{2\pi^2}} = \frac{11\pi^2}{216}$$

This should be equal to πR^2 where R is the average radius of a random circle in the unit cube

$$R\pi^2 = \frac{11\pi^2}{216}$$

$$R = \sqrt{\frac{11\pi}{216}} = 0.399986$$

Generating a random background and then plotting the radius histogram for this data set shows that there is indeed a peak around $R \approx 0.4$.

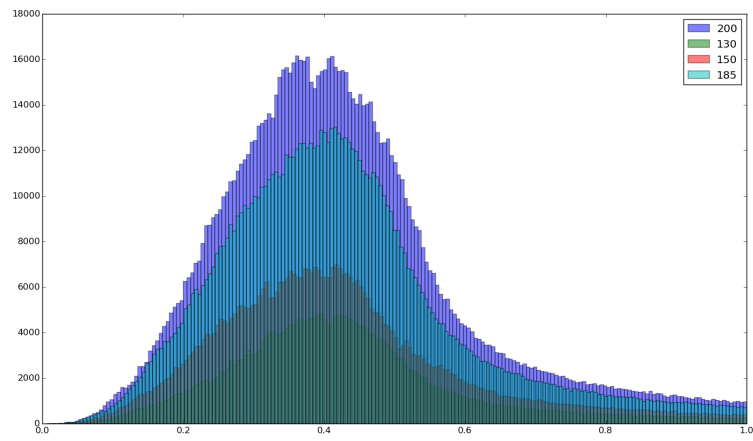


FIG. 3.13 – Radius distribution for uniformly generated background. The expected value of $R \approx 0.4$ is quite well represented for differently sized data sets

4 Results

In this section results for the conventional 1D, 2D, 3D Hough transform and the combinatorial triplet Hough transform are presented. 1D and 2D Hough transform were not pursued in depth. They are presented here as they offer a nice way of understanding how each extra dimension expands the algorithm and also shows the flaws with each added dimension.

For the 1D, 2D and 3D Hough transform very simple data was used. There were no physical constraints when generating the circles and the number of points per circle and the distribution of the radii doesn't reflect the real data obtained by LHCb.

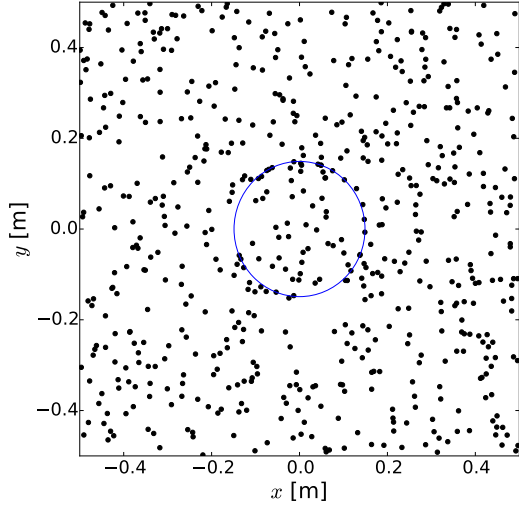
For these conventional Hough transforms following data sets were tested

- 1 circle and 600 background hits
- 2 circles and 0 background hits
- 5 circles and 30 background hits
- 6 circles and 200 background hits (results only shown for 2D and 3D)

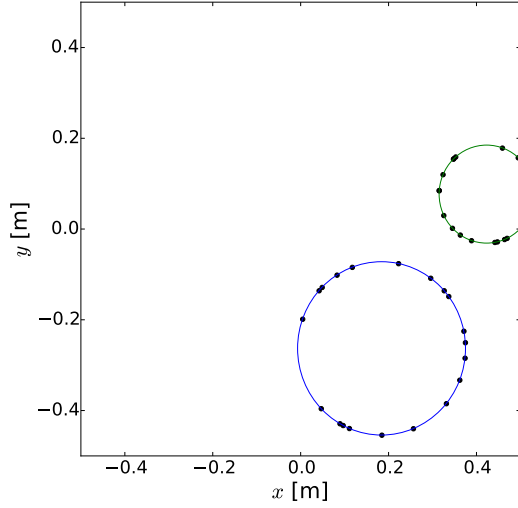
The points per circle ranges from 17–31 in all the circles from the data sets and the background is uniformly distributed in the unit square. The event with 1 circle and 600 background hits is meant to test how robust the algorithm is with a lot of background hits. The second test case is meant to test if the algorithm can handle two circle objects, the third is a mix between several circles and background and the last event is a special case where two circles lie very close to each other and for the 2D and 3D case where the center is unknown these algorithms can encounter problems with detecting the right circles. In Figure 4.1 the real circles plotted from simulation parameters for the different events.

4.1 1D Hough transform results

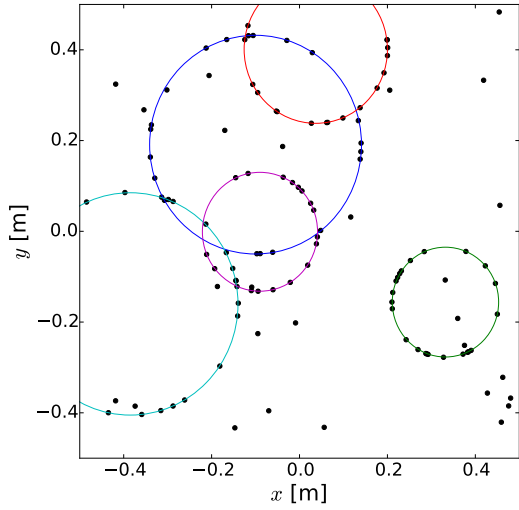
There are two different plots: for each set the radius score is shown in Figures 4.4 and 4.3 and the resulting circles in Figure 4.2. The radius score is a 1D plot of the weight function $w(\eta)$ 3.2. The highest peak indicating the maximum score and its location telling the value of the radius. The resulting circle plot is the center (which was known) and the extracted radius combined, drawing the resulting circle. The numerical values from the high scores and the second highest score are shown in Table 4.1. The algorithm has no problems to find any of the circles even with background. But this was expected as the algorithm only needs to search in one dimension and makes the whole process very easy. The fourth event was also tested for the 1D Hough transform but not shown in this thesis because they didn't add anything that is not already covered with the other 3 events.



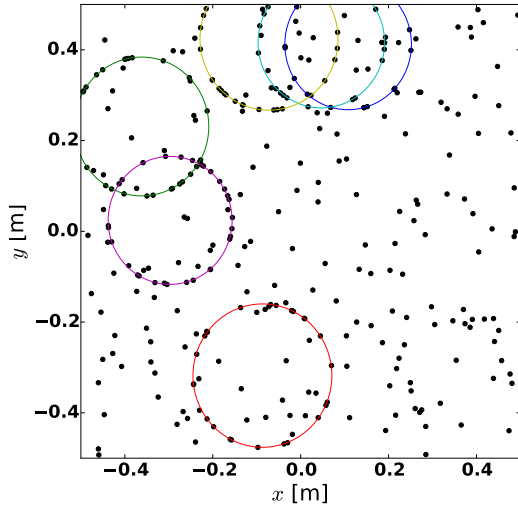
(A) 1 circle, 600 background hits.



(B) 2 circles, 0 background hits



(C) 5 circles, 30 background hits



(D) 6 circles, 200 background hits

FIG. 4.1 – These are the circles as generated by the simulation.

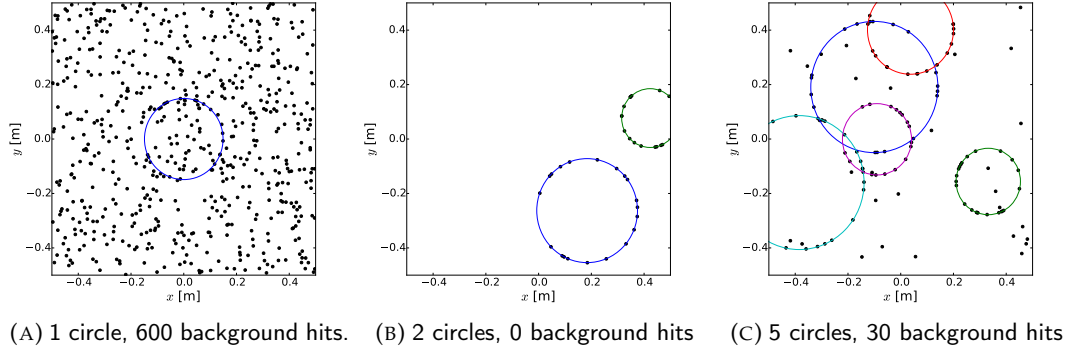


FIG. 4.2 – Circles found by the 1D Hough transform. The circle in Figure 4.2a has its center in the origin so the algorithm did find the circle

TAB. 4.1 – Radius scores for the different events. There is always a big difference between the highest score which determines the circle and the second highest score which is noise. In the example with 1 circle and 600 background the second highest score is relatively high which is due to the many background hits that coincidentally have the same center as the real circle but with a different radius than the real one

Event	Highest score	2nd highest score
1 circle 600 background hits	8'546	3'798
2 circle 0 background hits	7'904	828
	6'373	857
5 circle 30 background hits	7'519	1'401
	9'085	1'476
	6'758	1'341
	7'381	848
	7'410	1'498

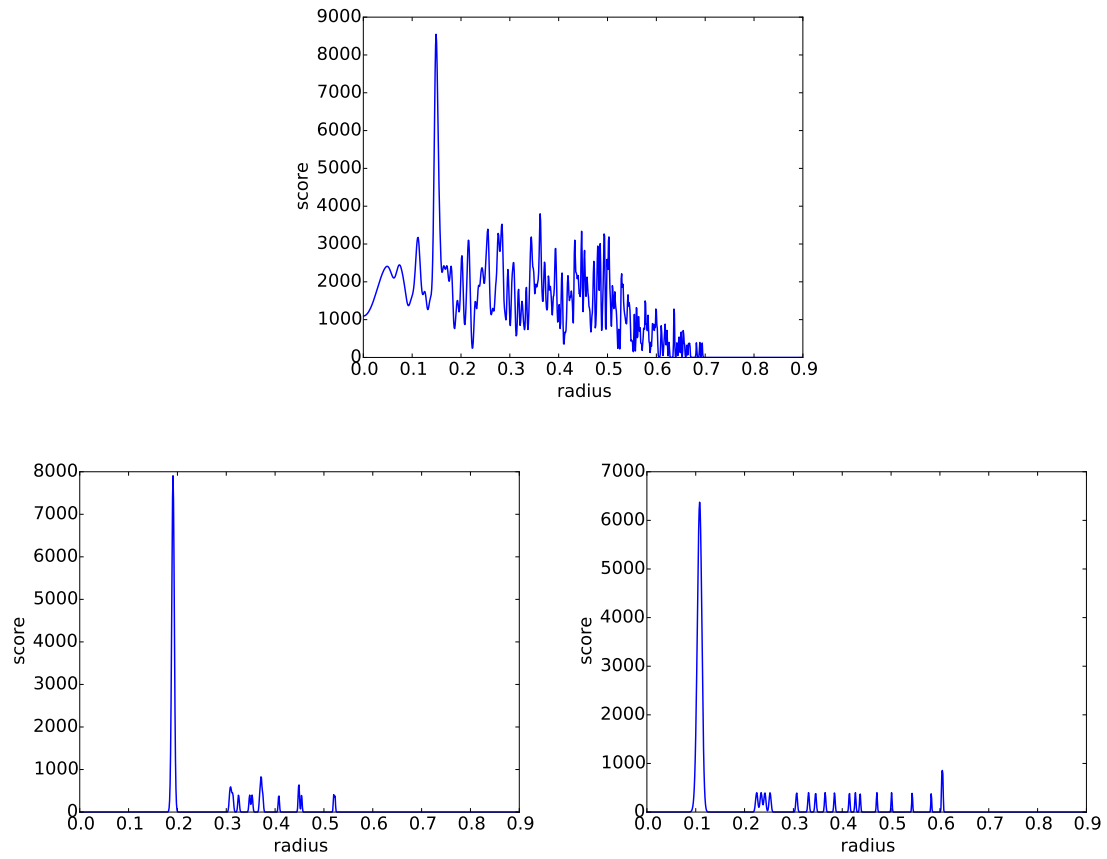


FIG. 4.3 – Radius scores for 1 circle and 600 background (top) and 2 circles with 0 background hits (bottom). For the event with only 1 circle and 600 backgrounds the scores from background hits for a given radius are higher than in the other events with less noise but the correct radius has still a distinct peak.

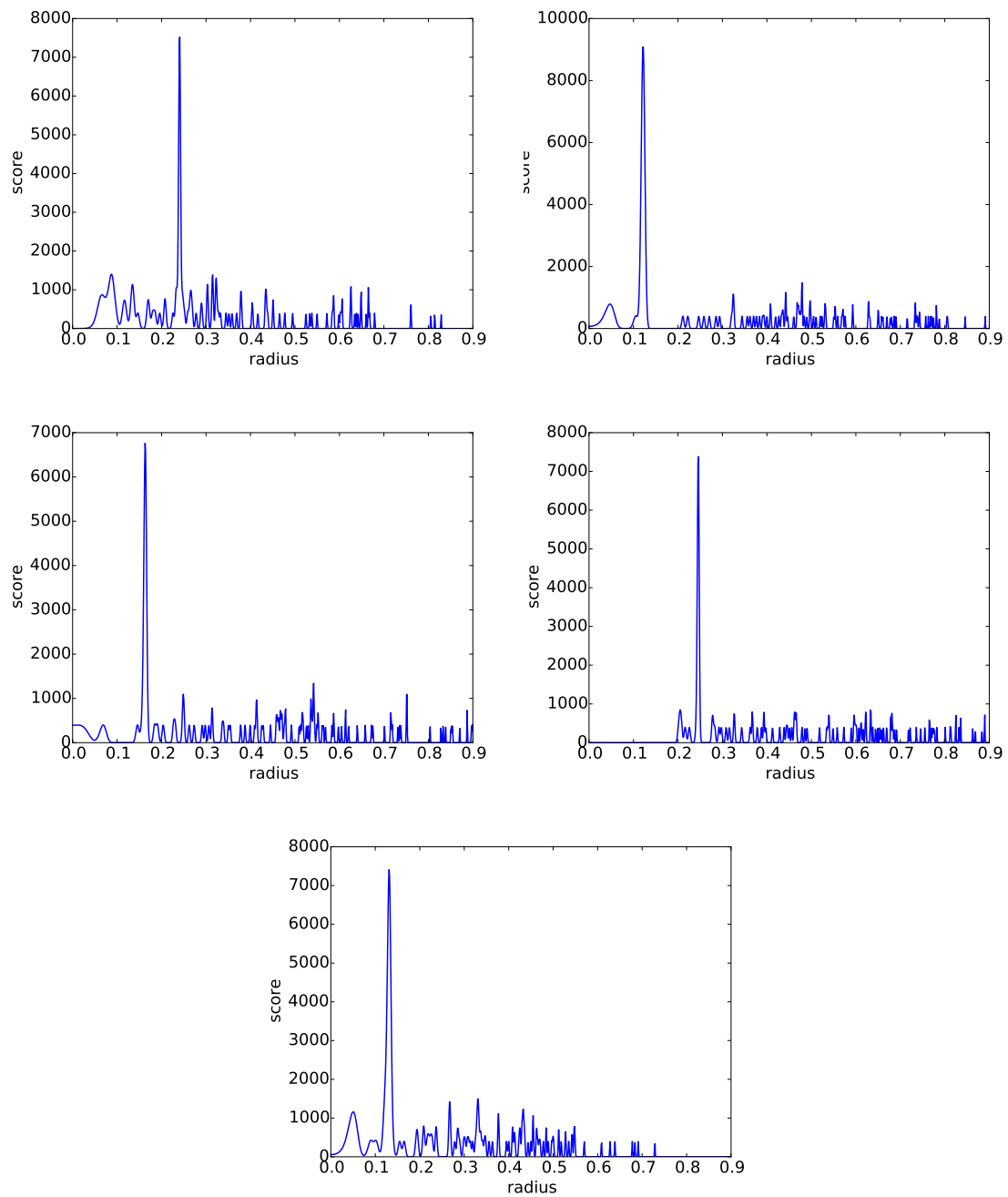


FIG. 4.4 – Radius scores for the 1D Hough transform for the 5 circles with 0 background.

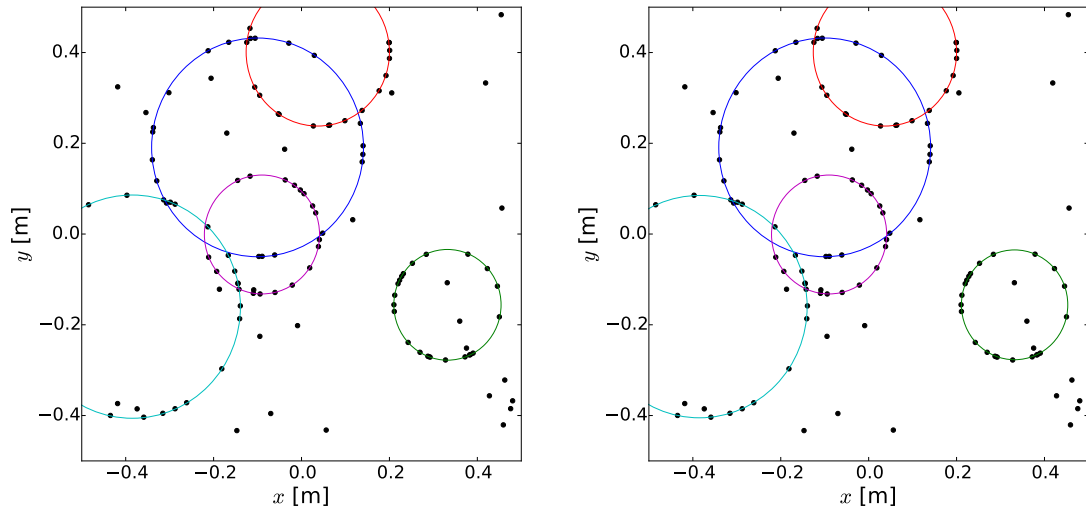


FIG. 4.5 – The result for 5 circles with 30 background hits. On the left the result obtained by the 1D Hough transform algorithm while the circles from the simulated data is on the right

TAB. 4.2 – Radius scores for the different events. There is always a big difference between the highest score which determines the circle and the second highest score which is noise. In the example with 1 circle and 600 background the second highest score is relatively high which is due to the many background hits that coincidentally have the same center as the real circle but with a different radius than the real one. For the event with 6 circles and 200 backgrounds sometimes the second highest score is extremely close to the highscore. This happens mainly when two circles have a similar radius and thus the second highest score is not just noise but actually another circle.

Event	Highest score	2nd highest score
1 circle 600 background hits	8720	5778
2 circles 0 background hits	7904	2154
	6373	2275
5 circles 30 background hits	7488	4489
	9095	3717
	6892	2387
	6498	2373
	6721	1953
6 circles 200 background hits	11869	5583
	10466	8687
	8762	5790
	6872	5203
	10155	3595
	4764	3759

4.2 2D Hough transform results

The 2D Hough transform uses again the weight function 3.2 to search for peaks with high score in the (x, y) for known radiuses. This means instead of a 1D score it is now a two dimensional plane where the highscore has to be found. Figure 4.9 shows how a slice out of this plane looks like (similar to the 1D radius histogram).

The same events as for the 1D Hough transform were investigated to make a comparison about reliability. Additionally the results for 6 circles and 200 background hits are shown in this section since the algorithm had issues with this event to detect all the circles initially.

The reason for the wrong detection of the circle is quite simple. Two of the circles have very similar radiuses. The algorithm looks first for the magenta circle (because that happens to be the way the centers are arranged in the list) and since so many points of the yellow circle lie so close together it is easy for the magenta circle (who has a similar radius) to get a high score with these points, a higher score than it would get with its proper points. If the yellow points were more evenly distributed on the circle or if there were more magenta points this probably wouldn't happen but that is something that can't be controlled.

In Figure 4.6 and 4.7 the circles found by the algorithm and in Table 4.2 the scores for the circle candidate and the second highest score are shown (see Figure 4.1 on page 33 for the results from the data).

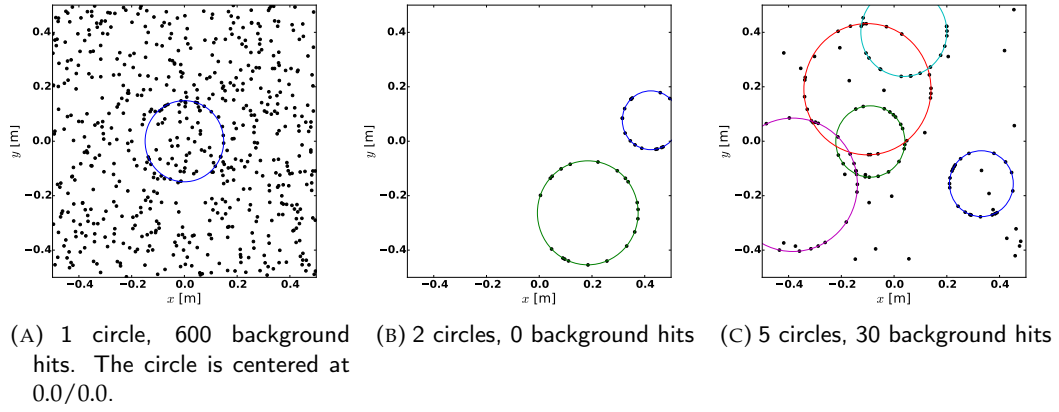


FIG. 4.6 – The first three events were solved correctly by 2D Hough transform

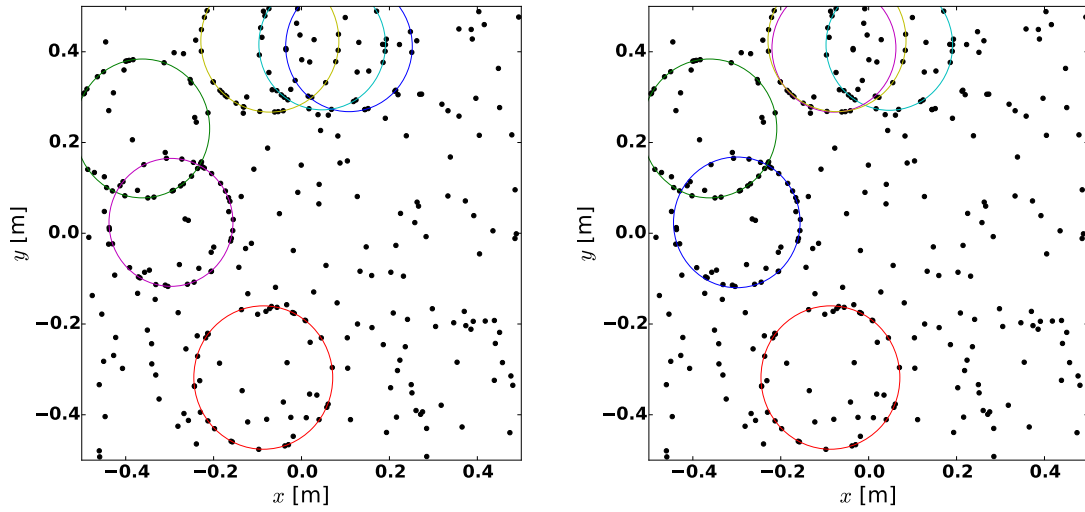


FIG. 4.7 – The rings generated by the simulation on the right and the rings found by the algorithm on the left. The magenta colored circle in the left image is incorrect. It replaced the yellow circle (taken from the simulated data on the left)

Theoretically it is also possible that the yellow circle gets fitted to the magenta points but since there are still enough yellow points left after the removal of the points assigned to the magenta circle they still have the highest score with their own points and the magenta circle goes unnoticed.

Also, if the yellow circle would be checked before magenta then the algorithm would find the proper circles as well. So it actually can depend on the order in which the circles are searched.

4.2.1 2D Hough transform, 2 circles, 0 background hits

Two circle objects have to be found in this event with zero background. This poses no problem unless the two circles have the same radius then without removing points that have been used for one circle it will always find the circle with the higher score. Introducing a mechanism that removes points that were assigned to a circle solves this problem.

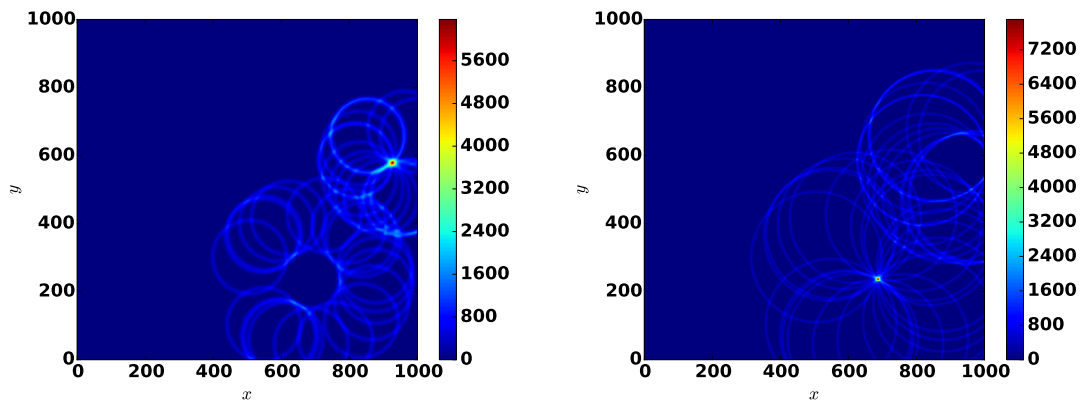


FIG. 4.8 – Center score for the 2D Hough transform for 2 circles with 0 background.

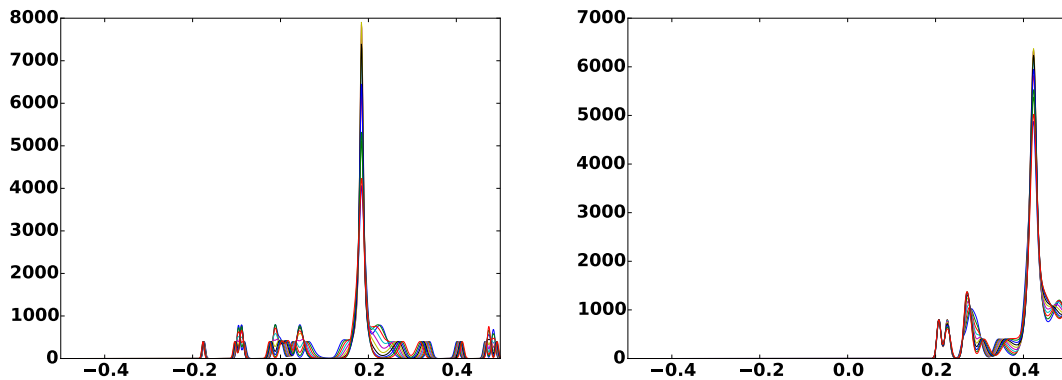


FIG. 4.9 – Each line in the figures is a horizontal slice around the maximum values out of the 2D histogram for the 2 circle event. These 2D histograms are very similar to the radius histogram from the 1D HT but now instead of the weights for the radius which is 1 dimensional the weights for x, y are calculated which are 2 dimensional

4.2.2 2D Hough transform, 5 circles, 30 background hits

This event added more circles but also some background hits. The algorithm had no problems finding all the circles. In Figure 4.10 the score matrix

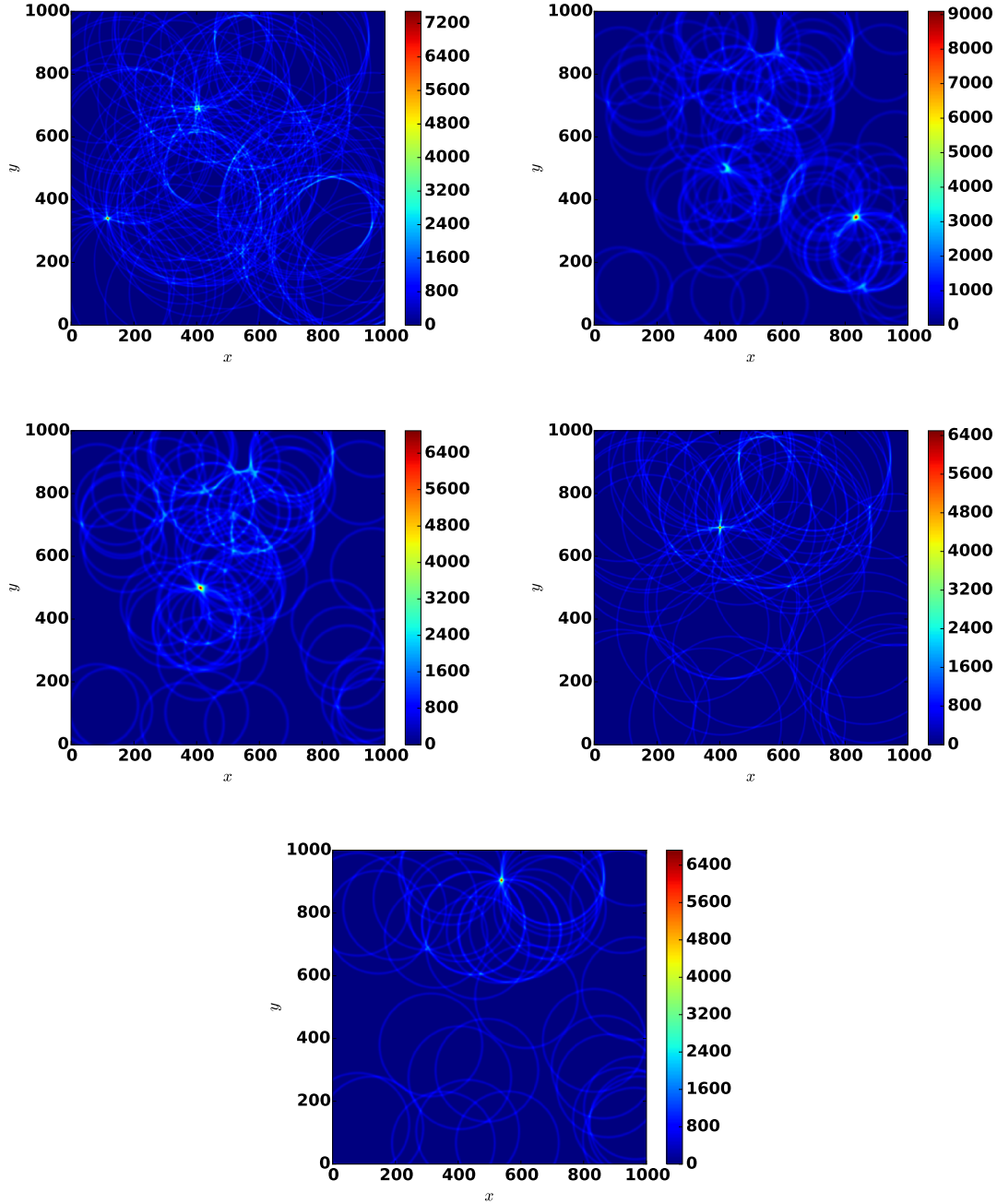


FIG. 4.10 – Center scores for all the centers for an event with 5 circles.

4.2.3 2D Hough transform, 6 circles, 200 background hits

As briefly discussed in the overview the algorithm does make mistakes as this event will show. 6 circles were generated with 200 background hits. The interesting part is not that amount of circles or the amount of background hits are the problem but the properties of

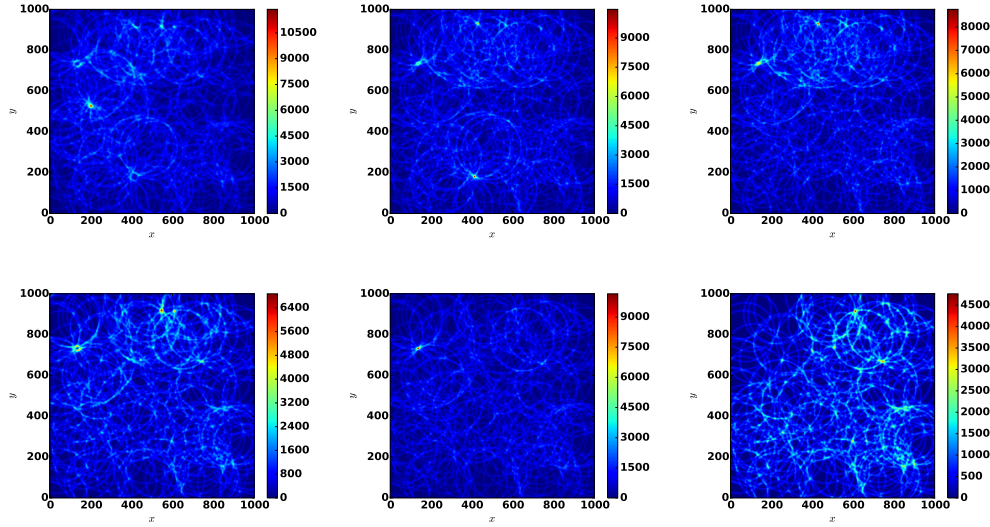


FIG. 4.11 – Center scores for 6 circles with 200 background hits. There is a lot more going on because of all the background hits that by accident contribute to a high score all over the grid.

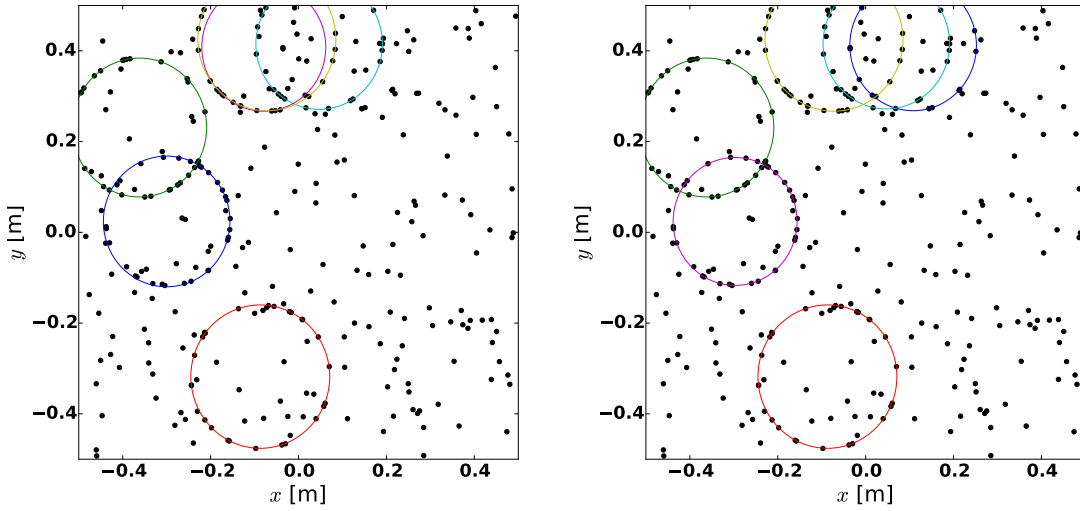


FIG. 4.12 – On the left side the wrong result obtained by the 2D Hough transform and the correct one on the right side

the circles (center coordinate and radius). In this example there is a misidentification of a circle with points of another.

A possible way to fix this particular problem is to tune the parameters of the weight function namely reducing the standard deviation.

$$w(\eta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\eta^2}{2\sigma^2}\right)$$

Having a higher σ means that a point that is a bit off of the circle still contributes a considerable value to the total score. The smaller the σ is the sharper the peak. However if the peak is too sharp then the algorithm might discard possible results because they are

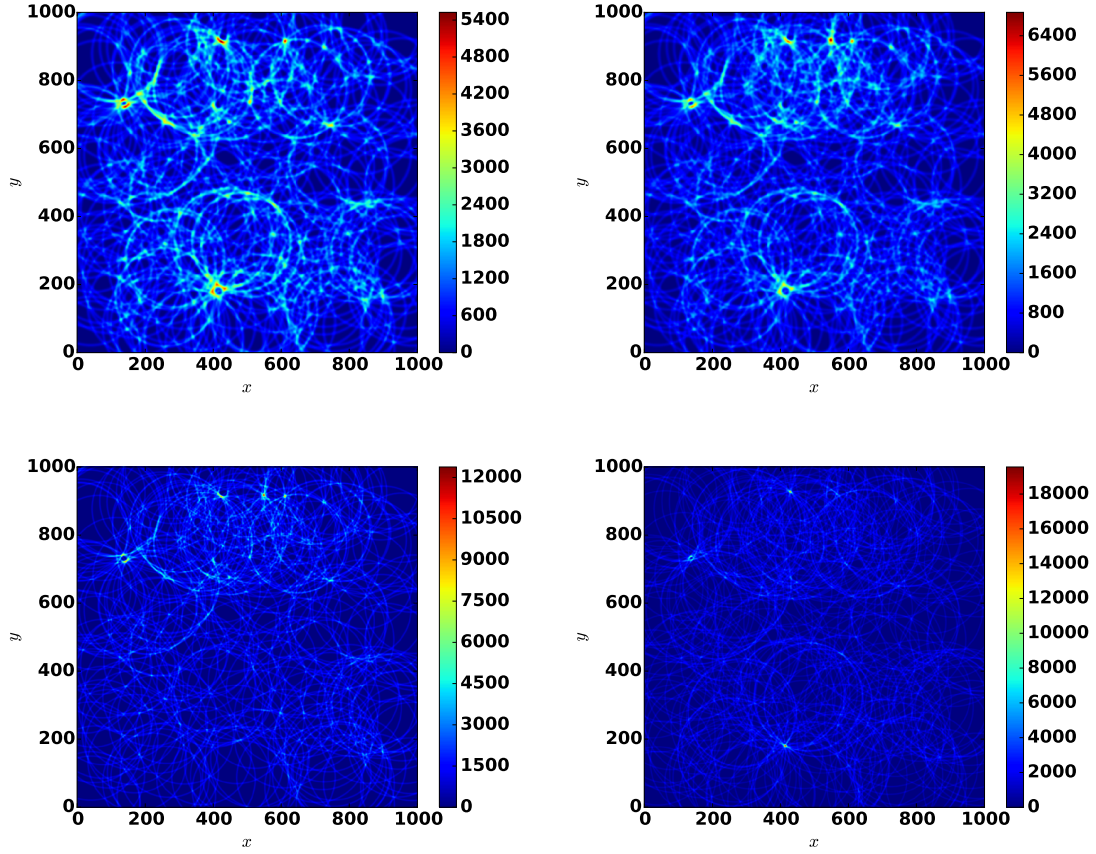


FIG. 4.13 – Old center score in the top row for circle 5 (magenta) and 6 (yellow). And below the same circles this time with the new $\sigma = 0.0005$. It is well visible how big the influence of sigma is for the center score. With a σ of 0.001 just by eye there seem to be many similar maxima whereas with a $\sigma = 0.0005$ the maximas get much more distinct and the maximum score is also higher. Where the high score in the top row is $\approx 5'400$ and $6'400$ in the bottom it is $\approx 12'000$ and $18'000$.

just a bit off the circle but since the peak is so narrow they don't contribute at all to the total score.

In the example before σ was equal to 0.001 while the space dimension was 1. So if the detector was 1 m per dimension a hit 1 mm off the perfect location contributes still more half of the score off the perfect location. If we set $\sigma = 0.0005$ a point 0.001 away from the perfect location it only contributes about 14% of the maximum score.

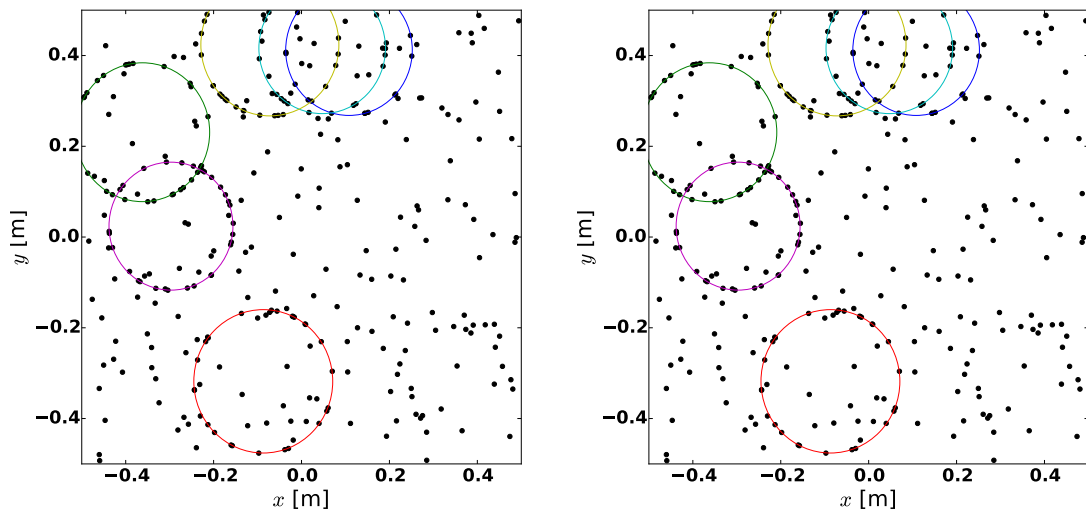
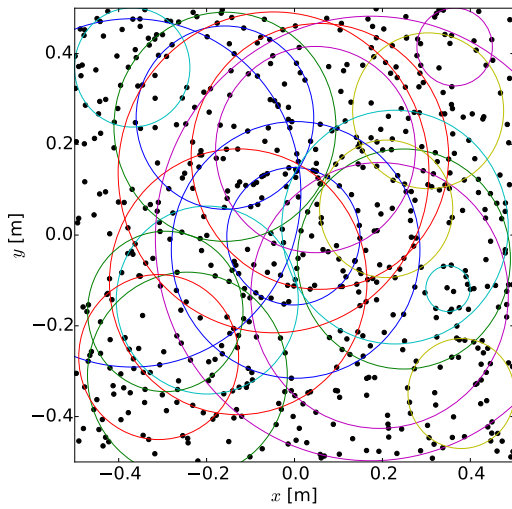


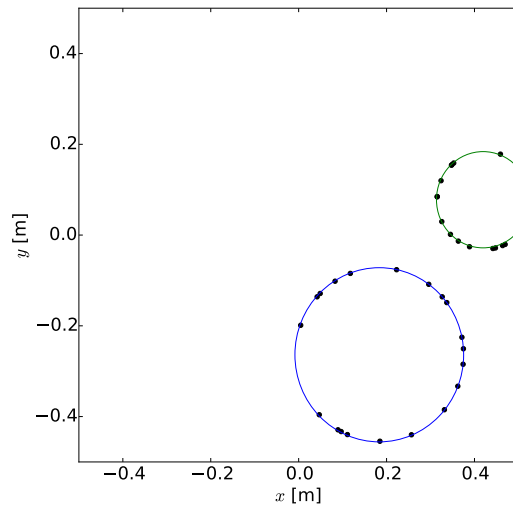
FIG. 4.14 – Again on the left the calculated result and the result taken from data on the right. With the new σ the algorithm is able to calculate all the circles correctly.

4.3 3D Hough transform Results

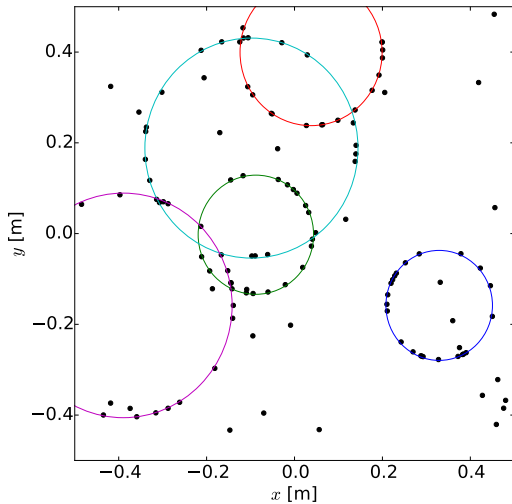
The 3D Hough transform has now to deal with 3 unknown parameters: x, y, r , which also means that the number of total circles is unknown. As explained in the code snippet 3.7 in Section 3.1.3 a threshold is needed to decide when the algorithm should stop finding a circle. This value depends heavily on the σ of the weight function. The σ was set to 0.001 and the threshold was set to 3500. For comparison again the same events as in the 1D and 2D Hough transform are studied to compare the reliability of the algorithms and how another unknown dimension adds to the complexity of finding circles. The plots of the found circle for the 4 events are shown in Figure 4.15.



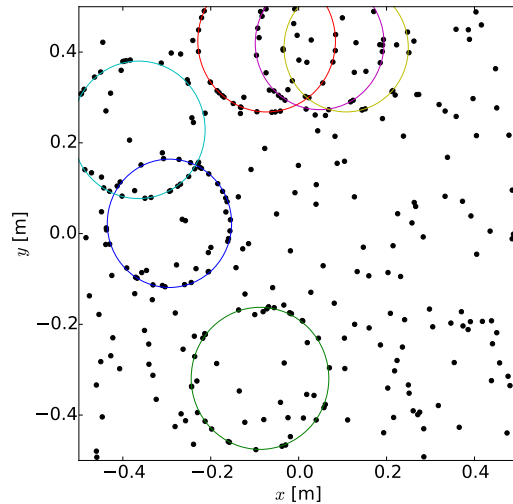
(A) 1 circle, 600 background hits. The circle is centered at 0.0/0.0.



(B) 2 circles, 0 background hits



(C) 5 circles, 30 background hits



(D) 6 circles, 200 background hits

FIG. 4.15 – Circles found by the 3D Hough transform. Figure 4.15a shows that with a low enough signal/noise ratio even this seemingly simple event fails to be calculated correctly. Even though the algorithm found the real circle it also found 22 others.

4.3.1 3D: Overview of the results

In general the 3D Hough transform works quite well. Even with the added complexity of an extra dimension it is capable to solve three of the four events correctly. One event that failed was the 1 circle and 600 background example where the signal to noise ratio was just too small for the algorithm to isolate the correct solution. The problem was that the threshold was set too low and thus it was unavoidable that the algorithm would fail. In Table ?? the scores for the 10 highest scores are shown. As a reminder, the threshold was set to 3500 initially. So the obvious thing seems to set the threshold to 6300 so the algorithm wouldn't pick up any fake rings. However, with a threshold that high legit rings would get lost. For a fixed threshold there will always be a case where rings get lost or fake rings get picked up. On the other hand the 3D Hough transform managed to solve the 6 circles with 200 background hits despite having less information. This is probably due to the fact that the 3D Hough transform finds the circles sorted by score and doesn't depend on the order in which circles are searched (remember, in the 2D case there is the radius given and depending on which radius is given first we can solve it correctly or not).

TAB. 4.3 – The ten highest scores for the case of 1 circle with 600 background hits.

8122	6256	5672	5654
5373	5290	5047	5000
4727	4647		

4.4 Combinatorial triplet Hough transform results

4.4.1 Overview of the results

This method was tested against a set of 10'000 events generated in a Monte-Carlo simulation with parameters (number of rings, number of points per ring, number of noise points) tuned to reproduce distributions measured in LHCb. The whole set was tested several times, each time with a different threshold for the radius and center histograms. Each of these runs used the speed up method of splitting the data point list into two lists and building triplets from the sublists. The results show that this choice yielded an efficiency above 98%. The performance was evaluated with an analysis script to measure efficiency and ghost rate (along with missed circles and fake circles).

The efficiency ϵ is defined as

$$\epsilon = 1 - \frac{m}{T} \quad (4.1)$$

where m is the number of circles that weren't matched by the algorithm and T the combined number of circles from all events. The ghost rate γ is defined as

$$\gamma = \frac{f}{T} \quad (4.2)$$

where f is the number of fake circles that were found by the algorithm but have no match in the test data and T again the total number of circles.

The main parameters for this algorithm are

- Histogram thresholds (radius and center histogram)

- Distance between centers of pairs of rings
- Difference between radiuses of pairs of rings

The last two parameters are only parameters for the analysis to decide whether or not the algorithm has found a circle comparing radius and center coordinates with the test data and if a circle is within these bounds the circle will be considered as found. In a real application however, these parameters are useless because a priori there is no knowledge about the true circles.

- Distance between the center of a calculated ring and the corresponding true ring
- Difference between calculated radius and the true radius of a ring

4.4.2 Removing duplicates

Without cleaning up the algorithm finds a lot of duplicates. As an illustration the 1D radius histogram where two almost adjacent bins have a high score as seen in Figure 4.16. They won't be directly adjacent because the algorithm searches for a high scoring bin and takes the left and right neighbour also into account and sets them to 0. It's still very likely that the two bins are from the same circle but they are treated now as independent because it was further than 1 bin away. The data extracted from the radius histogram also contains the center data. Now both scores are handled by the center extraction and there both have the maximum at the same x, y coordinate and the algorithm considers both as independent circles and without cleaning for duplicates the result can be seen in Figure 4.18

Following the code for extracting the radiuses (with their center data). So if bin i in the radius histogram H has a maximum we also save the list from `center[i]` to the corresponding radius. The function then returns a list of radiuses and for each radius in the list there is a list of center data which is used to extract the center of the circle.

```
edges #x edges of the histogram
center #center data
while True:
    i = max(H)
    n = NUMBER_OF_R_BINS
    n_entries = sum(H[i-1 if i>0 else i:i+2
                    if i<n-1 else i+1])
    if n_entries < RADIUS_THRESHOLD:
        # there are less than THRESHOLD
        # entries in 3 bins
        break
    radiuses.append(edges[i])
    index_list = range(i-1 if i>0 else i,i+2
                      if i<n-1 else i+1)
    for index in index_list:
        if center[index]:
            center_list += center[index]
        H[index] = 0
    center_data.append(center_list)
return radiuses, center_data
```

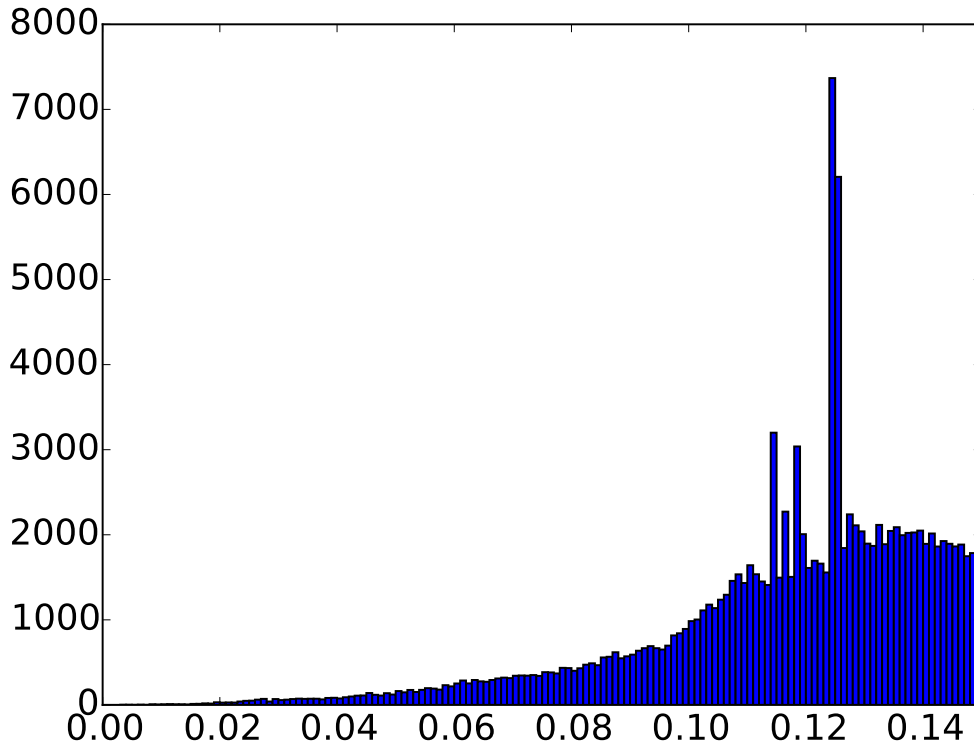


FIG. 4.16 – Two almost adjacent bins with a high score. Apart from the very clear peak of two bins next to each other at ≈ 0.125 with a score of 7000 and 6000 respectively, there are 3 smaller peaks separated by background bins. After picking up the highest peak at the score of 8000 and the neighbours left and right the next highest score is the one at roughly 3000. The algorithm also picks up the adjacent bins on the left and right but there are still peaks left that are above threshold.

As before once a maximum has been found and the sum of the maximum plus the adjacent neighbors is bigger than the threshold we have another candidate. The bin and the neighbors are then set to 0. This is repeated until the sum of a maximum plus the adjacent neighbors is smaller than the threshold, where the algorithm stops.

```
H #2d histogram with the center data
xedges
yedges
centers = []
n = NUMBER_OF_S_BINS
while True:
    i,j = max(H)
    score = sum(H[i-1 if i>0 else i:i+2
                if i<n else i+1,j]) +
            sum(H[i, j-1 if j>0 else
                j:j+2 if j+2 <= 3
                else j+1]) - H[i,j]
    if score < CENTER_THRESHOLD:
        break
    i_index = range(i-1 if i>0
```



```

        else i,i+2 if i<n else i+1)
j_index = range(j-1 if j>0
                else j,j+2 if j<n else j+1)
for ii in i_index:
    H[ii][j] = 0
for jj in j_index:
    H[i][jj] = 0
centers.append( {'center' : (xedges[i], yedges[j]),
                    'nEntries' : score } )

return centers

```

After this step the basic algorithm is done and in Figure 4.18 is the result. To get rid of this duplicates there is one more step. The algorithm now compares all the found circles among each other and if two circles are within a certain range center and radius wise they are considered to be duplicates and the one with a lower score gets discarded.

For this thesis the removal of the duplicates is done in the analysis part of the code so it was easier to tune the parameters.

```

res = []
sorted_results = sorted( results, key=lambda k:
                        k['nEntries'], reverse=True)
while len(sorted_results):
    circle = sorted_results.pop()
    unique = True
    for dic in sorted_results:
        if (np.linalg.norm(np.array(circle['center']) -
                                np.array(dic['center']))) <
            DUPLICATE_MAX_CENTER_DISTANCE and\
            (abs(circle['radius'] - dic['radius']) <
            DUPLICATE_MAX_RADIUS_DISTANCE):
            unique = False
            break
    if unique:
        res.append(circle)
return res

```

FIG. 4.17 – Pseudo code for removing possible duplicates from the circles found by the algorithm. First results are sorted by their number of entries in the histogram so the least relevant comes first. If for a given circle another circle exists with the same center and radius (within the cuts) then the circle is considered a duplicate and will be removed. `DUPLICATE_MAX_CENTER_DISTANCE` and `DUPLICATE_MAX_RADIUS_DISTANCE` are parameters that can be tuned

4.4.3 Different thresholds

The algorithm has a very simple threshold to decide whether a candidate is a circle or not. It is the number of hits in the radius and center histogram. This essentially puts a

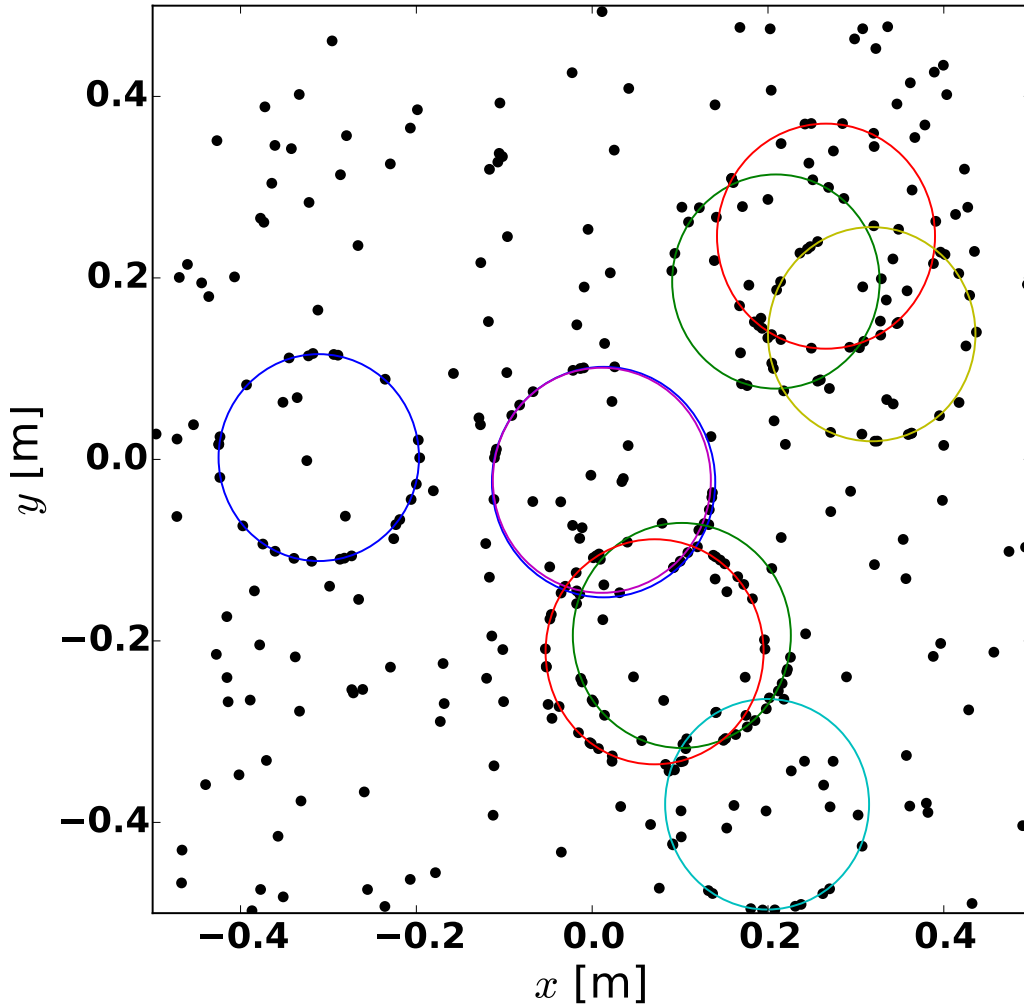


FIG. 4.18 – The result of the combinatorial triplet Hough transform before cleaning up the duplicates. Blue and purple are basically the same circle.

constraint on the minimum amount of points a circle should have and the threshold then is of course the binomial coefficient of that number

$$\binom{\text{Threshold}}{3}$$

In table 4.4 the different efficiencies for the algorithm with different thresholds can be seen. Intuitively one expects to have more fake detections at a lower threshold since it is more likely for some random hits to form a circle. Starting from a certain threshold the efficiency will fall again when legit circles will fall under the threshold but analysing the test data set shows that approximately 98% of the circles have more than 12 points per circle.

This gives the algorithm a upper boundary for the efficiency (lower boundary of how many circles won't be found). No matter how accurate the algorithm is, if it doesn't consider circles below the threshold it will never find these. Thus the higher the needed number of points per circle is, the lower the efficiency will be. But the threshold is not necessarily the only factor for the lower boundary. Depending on how the radius and

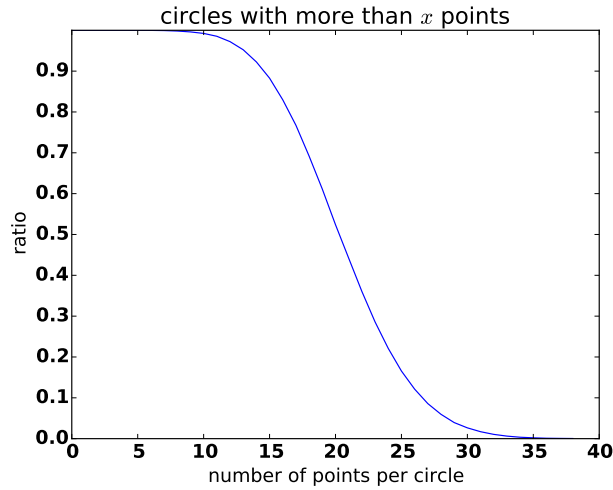


FIG. 4.19 – Reversed cumulative distribution of the points per circle. The plot shows the ratio of circles with more than x points over the total number of circles

TAB. 4.4 – Efficiencies for different thresholds (radius and center cuts are fixed at 0.006). The ghost efficiency explodes if the threshold is too low, that is when combinations from background/background, background/circle or circleA/circleB points classify as a circle

Threshold	ϵ	Missed	γ	Fakes
20	99.99%	7	183.22%	91573
35	99.98%	9	131.50%	65723
56	99.97%	16	9.07%	4533
84	99.92%	41	2.13%	1067
165	99.57%	214	0.13%	66
220	99.18%	410	0.05%	27
286	98.43%	784	0.01%	4

center cut is chosen - more circles can get lost because they get identified as duplicate when they are in fact just another distinct circle in the vicinity of another circle.

4.4.4 Different cuts for circle selection

Setting the histogram threshold is just the first step to finding the circles. The next step is removing duplicate circles that are created with a major fraction of points of a circle and for example a single background hit that is just far enough off that the radius and center histogram show it as a distinct circle. Depending on how loose this cut is, there is a chance that an actual circle gets discarded as a duplicate if their centers happen to be very close to each other and they have very similar radiuses. For both to happen simultaneously seems rather unlikely so the loss of circle due to this cut should be rather low.

Cut variation for a fixed threshold

In tables 4.5 and 4.6 different cut parameters were tested and compared for the efficiency, ghost rate, duplicate and missed circles.

TAB. 4.5 – Reducing ghost rate for a fixed radius cut (0.003) with varying center cut. The tighter the center cut is the more duplicate circle remain after the

C cut	ϵ	Missed	γ	Fakes
0.003	99.19%	405	0.17%	86
0.006	99.19%	407	0.12%	62
0.009	99.18%	409	0.12%	60
0.012	99.17%	415	0.12%	60

TAB. 4.6 – Reducing ghost rate for a fixed center cut (0.003) with varying center cut. The tighter the center cut is the more duplicate circles fail being detected by the removeDuplicate code shown above

R cut	ϵ	Missed	γ	Fakes
0.003	99.19%	405	0.17%	86
0.006	99.19%	405	0.15%	77
0.009	99.19%	405	0.15%	77
0.012	99.19%	405	0.15%	76

What both tables show that there is only a limited sensitivity to a single parameter change. So in the following tables there is all the possible combinations for different cuts for both radius and center ranging from 0.003 to 0.0012 in steps of 0.003 for different thresholds.

TAB. 4.7 – Results for all the cut combinations for radius and center cuts

Threshold = 35 (7 points for a circle)					
<i>R cut</i>	<i>C cut</i>	ϵ	Missed	γ	Fakes
0.003	0.003	99.99%	4	178.96%	89444
0.003	0.006	99.99%	6	151.62%	75777
0.003	0.009	99.98%	8	144.59%	72264
0.003	0.012	99.97%	13	140.88%	70408
0.006	0.003	99.99%	4	174.01%	86968
0.006	0.006	99.98%	9	131.50%	65723
0.006	0.009	99.96%	19	118.91%	59432
0.006	0.012	99.94%	28	113.54%	56745
0.009	0.003	99.99%	4	173.53%	86729
0.009	0.006	99.98%	12	130.54%	65244
0.009	0.009	99.95%	25	112.21%	56079
0.009	0.012	99.93%	36	103.50%	51727
0.012	0.003	99.99%	4	173.30%	86612
0.012	0.006	99.97%	15	130.26%	65103
0.012	0.009	99.94%	32	111.64%	55795
0.012	0.012	99.91%	47	99.85%	49902
Threshold = 56 (8 points for a circle)					
<i>R cut</i>	<i>C cut</i>	ϵ	Missed	γ	Fakes
0.003	0.003	99.98%	11	20.54%	10264
0.003	0.006	99.97%	13	14.52%	7255
0.003	0.009	99.97%	15	13.59%	6792
0.003	0.012	99.96%	21	13.24%	6616
0.006	0.003	99.98%	11	19.23%	9611
0.006	0.006	99.97%	16	9.07%	4533
0.006	0.009	99.95%	26	7.29%	3641
0.006	0.012	99.93%	35	6.82%	3409
0.009	0.003	99.98%	11	19.14%	9568
0.009	0.006	99.96%	19	8.91%	4454
0.009	0.009	99.94%	32	5.90%	2950
0.009	0.012	99.91%	43	4.94%	2467
0.012	0.003	99.98%	11	19.10%	9544
0.012	0.006	99.96%	22	8.87%	4432
0.012	0.009	99.92%	39	5.81%	2906
0.012	0.012	99.89%	54	4.29%	2143
Threshold = 84 (9 points for a circle)					
<i>R cut</i>	<i>C cut</i>	ϵ	Missed	γ	Fakes
0.003	0.003	99.93%	36	5.82%	2911
0.003	0.006	99.92%	38	4.00%	1999
0.003	0.009	99.92%	40	3.80%	1900
0.003	0.012	99.91%	46	3.72%	1861
0.006	0.003	99.93%	36	5.37%	2682
0.006	0.006	99.92%	41	2.13%	1067
0.006	0.009	99.90%	51	1.66%	830
0.006	0.012	99.88%	60	1.57%	787

0.009	0.003	99.93%	36	5.34%	2671
0.009	0.006	99.91%	44	2.09%	1044
0.009	0.009	99.89%	57	1.27%	636
0.009	0.012	99.86%	68	1.07%	534
0.012	0.003	99.93%	36	5.33%	2665
0.012	0.006	99.91%	47	2.08%	1040
0.012	0.009	99.87%	64	1.25%	626
0.012	0.012	99.84%	79	0.88%	439

Threshold = 165 (11 points for a circle)

<i>R</i> cut	<i>C</i> cut	ϵ	Missed	γ	Fakes
0.003	0.003	99.58%	209	0.48%	238
0.003	0.006	99.58%	211	0.32%	162
0.003	0.009	99.57%	213	0.31%	156
0.003	0.012	99.56%	219	0.31%	155
0.006	0.003	99.58%	209	0.42%	209
0.006	0.006	99.57%	214	0.13%	66
0.006	0.009	99.55%	224	0.10%	50
0.006	0.012	99.53%	233	0.10%	49
0.009	0.003	99.58%	209	0.42%	209
0.009	0.006	99.57%	217	0.13%	64
0.009	0.009	99.54%	230	0.07%	34
0.009	0.012	99.52%	241	0.05%	26
0.012	0.003	99.58%	209	0.42%	208
0.012	0.006	99.56%	220	0.13%	64
0.012	0.009	99.53%	237	0.07%	34
0.012	0.012	99.50%	252	0.04%	18

Threshold = 220 (12 points for a circle)

<i>R</i> cut	<i>C</i> cut	ϵ	Missed	γ	Fakes
0.003	0.003	99.19%	405	0.17%	86
0.003	0.006	99.19%	407	0.12%	62
0.003	0.009	99.18%	409	0.12%	60
0.003	0.012	99.17%	415	0.12%	60
0.006	0.003	99.19%	405	0.15%	77
0.006	0.006	99.18%	410	0.05%	27
0.006	0.009	99.16%	420	0.04%	21
0.006	0.012	99.14%	429	0.04%	21
0.009	0.003	99.19%	405	0.15%	77
0.009	0.006	99.17%	413	0.05%	25
0.009	0.009	99.15%	426	0.02%	11
0.009	0.012	99.13%	437	0.02%	9
0.012	0.003	99.19%	405	0.15%	76
0.012	0.006	99.17%	416	0.05%	25
0.012	0.009	99.13%	433	0.02%	11
0.012	0.012	99.10%	448	0.01%	6

Threshold = 286 (13 points for a circle)

<i>R</i> cut	<i>C</i> cut	ϵ	Missed	γ	Fakes
0.003	0.003	98.44%	779	0.05%	25

0.003	0.006	98.44%	781	0.03%	17
0.003	0.009	98.43%	783	0.03%	16
0.003	0.012	98.42%	789	0.03%	16
0.006	0.003	98.44%	779	0.04%	21
0.006	0.006	98.43%	784	0.01%	4
0.006	0.009	98.41%	794	0.01%	3
0.006	0.012	98.39%	803	0.01%	3
0.009	0.003	98.44%	779	0.04%	21
0.009	0.006	98.43%	787	0.01%	4
0.009	0.009	98.40%	800	0.00%	2
0.009	0.012	98.38%	811	0.00%	2
0.012	0.003	98.44%	779	0.04%	21
0.012	0.006	98.42%	790	0.01%	4
0.012	0.009	98.39%	807	0.00%	2
0.012	0.012	98.36%	822	0.00%	2

These results show that the biggest influence for changing the ghost rate is adapting the threshold for the combinations. If the threshold is too low, too many contributions come from random point combination that happen to lie on some circle.

The main reason for missed circles is again the threshold since it limits the number of circles being found as circles only if their number of points is greater or equal than the threshold. And since the algorithm uses split lists the number of missed circles is even slightly higher due to the fact that sometimes the points of a circle can be split just in a way that the two lists don't contain enough points individually to create an accepted circle (see subsection 4.4.3).

5 Conclusions

This thesis was studying different Hough transforms for circle detection. In a first approach a 1D Hough transform was used and then extended to two and three dimensions. As a fourth and final approach a new method was developed for the circle detection.

The 1D Hough transform is a very stable method having no troubles to find different circles in different kinds of set ups. Even a high background doesn't impede the accuracy of the method. It is also a very fast since it only has to search one dimension for a circle. This approach is similar to the one used at LHCb (although a lot simpler presented here in this thesis).

The 2D Hough transform shows a good performance as well but it suffers from lower speed due to the increased complexity namely searching 2 dimension for the parameters instead of just one. Another problem is also the misidentification of circles based on the order in which they are searched for and also the sensitivity of the weight function used to weigh the centers for a given data points and radius. As the 1D Hough transform this is a very simplistic implementation and should have a lot of potential to improve, one improvement being to limit the range of where a potential center is searched for based on cuts (e.g. a center cannot have a radius > 0.30 which then would reduce the amount of grid points that have to be weighed per data point).

The 3D Hough transform has another dimension added since now is nothing known about the circles and it shows in the results. The main weakness of this method is the score threshold which tells the algorithm when to stop looking for a circle (the 1D and 2D transform didn't have this problem since due to the information given it was known, how many circles there were). If the threshold is too low, too many background hits can form a circle - if it's too high a valid circle might get lost. The difficulty in defining a score threshold is that it is based on the scoring function 3.2. Depending on the σ chosen the scoring function can behave quite differently. A problem related to that is that the 3D Hough transform strongly depends on the bin size of the parameter space. The bins shouldn't be too big or the algorithm becomes inaccurate because the scoring function uses inaccurate values. If the bins are too small, the algorithm takes forever due to the $\mathcal{O}(N^3)$ complexity. But the disadvantage can also have advantages because now the algorithm doesn't depend anymore on the order of which it looks for circles - it always picks the strongest candidates first and every subsequent candidate has a lower score until the score is so low that it's unlikely that it is a circle.

Finally the core component of this thesis the combinatorial triplet Hough transform. It is based on the fact that 3 points define a circle. Given 3 points in a 2D space it is possible to calculate the center and radius of the circle on which these 3 points lie on. Given a list of points a list of all possible triplets can be used to find possible circles. The algorithm works very well and maintains a high efficiency with all tested parameters but suffers as the 3D Hough transform from a high runtime (up to several minutes). However given the high accuracy of the algorithm (binning has almost no effect) makes this approach still very fascinating and offers maybe some potential for the future. The parameter with the biggest impact is the threshold which defines how many points need to lie on a circle

in order to qualify as a circle candidate. Choosing this threshold too low many fake circles appear and the ghost rate skyrockets. Choosing this threshold too high means an inherent loss of efficiency from which the algorithm can never recover since circles with a point count lower than the threshold will never be considered as a candidate.

Bibliography

- [1] Georges Aad et al. “The ATLAS experiment at the CERN large hadron collider”. In: *Journal of Instrumentation* 3.8 (2008).
- [2] Kenneth Aamodt et al. “The ALICE experiment at the CERN LHC”. In: *Journal of Instrumentation* 3.08 (2008), S08002.
- [3] Augusto Alves Jr et al. “The LHCb detector at the LHC”. In: *Journal of instrumentation* 3.08 (2008), S08005.
- [4] CMS Collaboration, R Adolphi, et al. “The CMS experiment at the CERN LHC”. In: *Jinst* 3.08 (2008), S08004.
- [5] LHCb Collaboration. *The LHCb Detector*. Online. URL: <http://lhcb-public.web.cern.ch/lhcb-public/en/Detector/Detector-en.html>.
- [6] A Powell. *Particle identification at LHCb*. Tech. rep. 2011.
- [7] TDR LHCb. *RICH technical design report*. Tech. rep. CERN-LHCC-2000-037, 2000.
- [8] RICH LHCb. “Technical Design Report”. In: *CERN/LHCC* 37 (2000), p. 2000.
- [9] Richard O. Duda and Peter E. Hart. “Use of the Hough Transformation to Detect Lines and Curves in Pictures”. In: *Commun. ACM* 15.1 (Jan. 1972), pp. 11–15. ISSN: 0001-0782. DOI: 10.1145/361237.361242. URL: <http://doi.acm.org/10.1145/361237.361242>.
- [10] H.P.V. C. *Method and means for recognizing complex patterns*. US Patent 3,069,654. Dec. 1962. URL: <https://www.google.com/patents/US3069654>.
- [11] M. Adinolfi et al. “Performance of the LHCb RICH detector at the LHC”. In: *Eur. Phys. J. C* 73 (Nov. 2012), p. 2431. DOI: 10.1140/epjc/s10052-013-2431-9. eprint: 1211.6759.
- [12] Roger Forty, LHCb Collaboration, et al. “RICH pattern recognition for LHCb”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 433.1 (1999), pp. 257–261.
- [13] Christian Blatter. *The expected area of a triangle formed by three points randomly chosen from the unit square*. Online. Apr. 2015. URL: <http://math.stackexchange.com/questions/1236958/the-expected-area-of-a-triangle-formed-by-three-points-randomly-chosen-from-the>.
- [14] Eric W. Weisstein. *Circle Triangle Picking*. Online. URL: <http://mathworld.wolfram.com/CircleTrianglePicking.html>.