Abstract

This thesis studies an algorithms for the detection of circles which are produced by particles travelling through the RICH detectors generating Cherenkov radiation. These circles can be detected with the help of the Hough transform. There is a short introduction to the linear Hough transform (which detects 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, the 1D, 2D and 3D Hough transform. The 1D Hough transform is used when the center is given and the radius needs to be found. The 2D Hough transform looks for circle centers ((x, y) coordinate) for a given radius. The 3D Hough transform is used when neither center nor radius are known. This is also the case for the data from LHCb where there are data points (signal and noise) 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 circle center we are intersted int) and the radius of the circumcircle.

Contents

List of Tables

List of Figures

1 Introduction

1.1 LHC - Large Hadron Collider

The Large Hadron Collider (LHC) is a proton-proton collider. It is the largest and highest-energy particle accelerator in the world. 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, and in particular for the search of the Higgs boson (which has been confirmed this year) and signs for new physics beyond the Standard Model of particle physics. The LHC lies in a tunnel 27 km in circumference and 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 core physics programm for the period 2014-2015 includes:

- <COMMENT: PROGRAM>

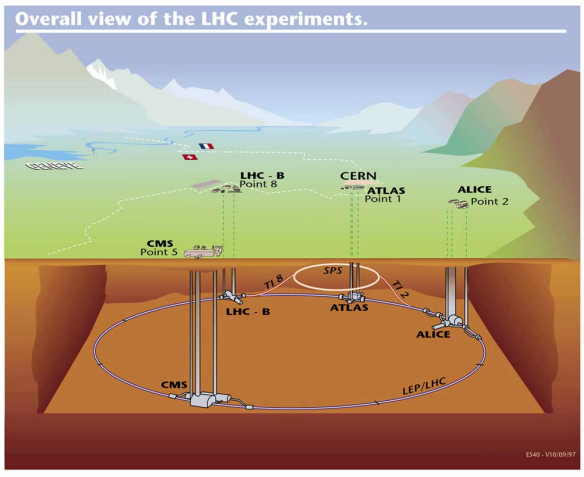


FIG. 1.1 – The LHC ring with its 4 experiments: ATLAS, CMS, Alice and LHCb

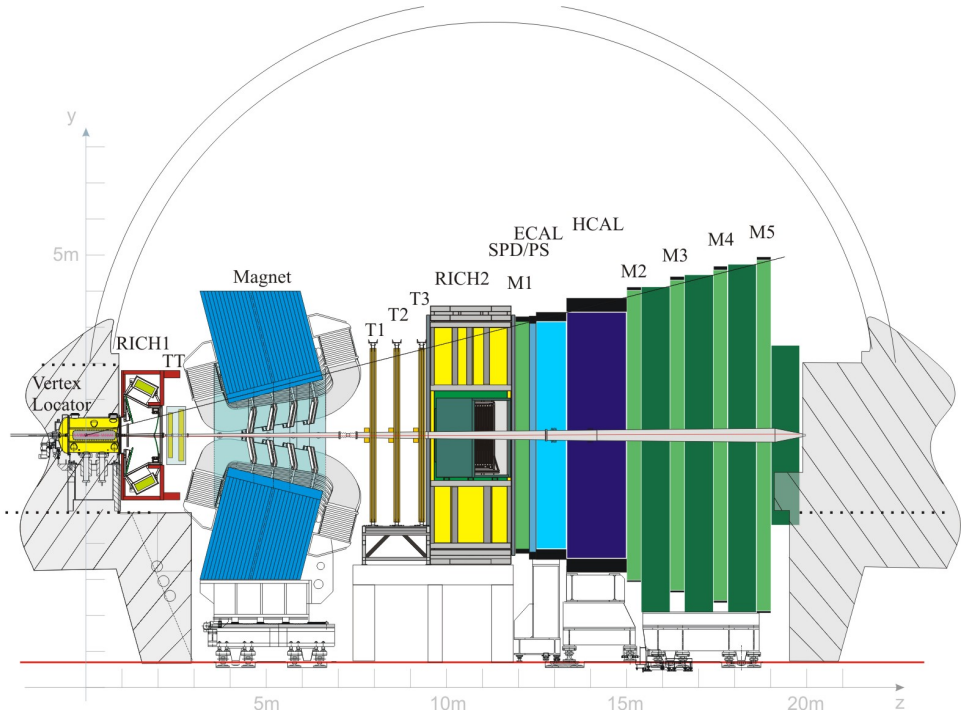


FIG. 1.2 – LHCb Detector: RICH1 before the magnet and RICH2 after the magnet with silicon strip detector in between and muon and calorimeter at the end.

1.2 LHCb

The LHCb is one of the four big experiments conducted at the LHC (ATLAS, CMS and ALICE being the other 3). The aim of this experiment is the study of decays of particles containing b and $\bar{b}$ particles (B-Mesons). During collisions these particles don't fly in all direction but rather stay close to the beam pipe. This is reflected in the design of the LHCb detector which is a foward arm spectrometer 20 meters long with subdetectors along the beam pipe as seen in figure 1.2.

A quick overview of the detector parts [2].

VELO The VErtex LOcator is where the beams collide and $b/\bar{b}$ are produced. The VELO measures the distance between the photon collision point and the point where B particles decay. Thus the B particles are not measured directly but inferred from the separation of these two points.

RICH The RICH detectors are built for particle identification. 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 emmited if a particle travels faster than the speed of light through a certain medium (often compared to breaking the sound barrier).

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

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

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 large tracking stations, each covering about $40\,\mathrm{m}^2$. It is used to determine the position where the particle passed the detector. In the silicon detector a charge gets deposited on a strip which defines the position and in the gas-filled tubes of the outer tracker a passing particle ionizes the gas molecules producing electrons.

Calorimeters They are designed to stop particles and measing the amount of energy lost while coming to a halt. The design of the stations is sandwich like. One metal plate and one plastic plate. A particle hitting the metal plate causes a secondary shower which induces a UV light in the plastic plate. The energy lost is proportional to the amount of light emmitted. It is the main way of identifying particles with no charge (e.g. photons, neutrons).

Muon system Muons are present in the final stage of B decays and thus important for the LHCb experiment. There are 5 rectangular stations increasing in size at the end of the detector. The total area covered by these stations is about $435\,\mathrm{m}^2$. Each station is filled with a combination of 3 gases. Passing muons react with this mixture and wire electrodes detect the result.

1.2.1 Particle identification

An important requirement at LHCb is the 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 indentification of said particles. The Muon system identifies muons to a very high level of purity which is esential 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 outisde the detector acceptance [5].

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 can 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. Moreover, the velocity that must be exceeded is the phase velocity (v_{Phase} or short v_{P}) and not the group velocity ($v_{\text{Group}} = \frac{\partial\omega}{\partial k}$).

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

As a charged particle travels through the medium, it disrupts the local electromagnetic field. If the particle travels slowly then the disturbance elastically relaxes to the mechinal equilibrium as the particle passes. However, if the particle travels fast enough, 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 coherent shockwave.

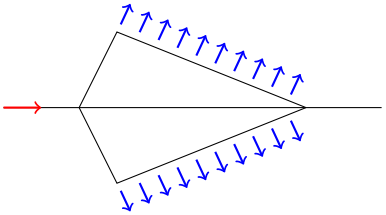


FIG. 2.1 – Cherenkov radiation

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

which is independent from the angle θ.

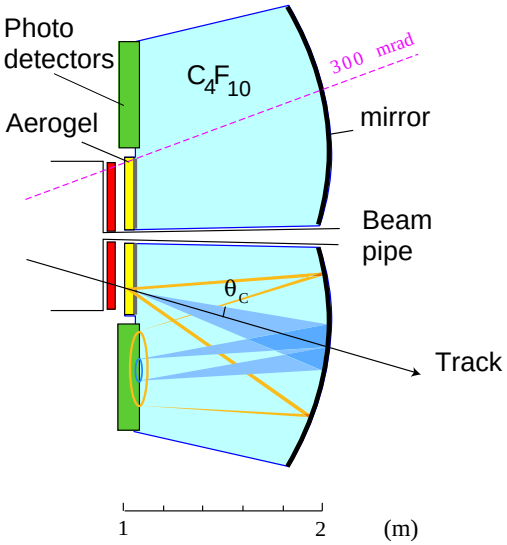


FIG. 2.2 – RICH-1 detector [4].

2.2 RICH detector

Particle identification is a fundamental requirement at the LHCb experiment. Meaningful CP-violation measurements are only possible if hadron identification is available hence the ability to distinguish between kaons and pions is essential. The LHCb experiment is unique in the sense that its hadronic particle identification is handled only by the RICH sub-detectors. This means – as mentioned before – that the RICH has to cover a wide range of momentum (1-100 GeV/c).

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} which covers the intermediate region of momentum. For the highest momentum tracks, gaseous CF_4 is used in the RICH-2.

There is a strong corelation between the polar angle and momentum of the tracks. Tracks with a wider angle often 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 acceptenace by the magnet.

Both sub-detectors are located in low magnetic field regions to keep the tracks straight while they pass through the radiators.

2.3 Hough transform

The Hough transform 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 cass 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 accumlator space that is explicitly constructed by the algorithm for computing the Hough transform.

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

2.3.1 Linear Hough transform

A linear function is normally defined as the following:

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

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 ρ-θ parametrization [3].

$$r = x \cos\theta + y \sin\theta$$

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

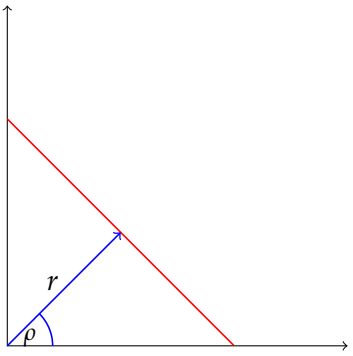


FIG. 2.3 – ρ-θ parametrisation

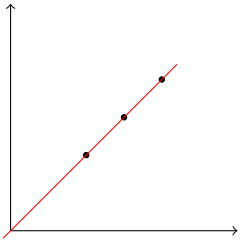
This means given a single point in the plane, the set of all lines going through this point form a sinusoidal curve in ρ-θ space. Another point that lies on the

TAB. 2.1 – Angle vs Distance

Angle	Distance
0	2.334

same straight line in the plane will produce a sinusoidal curve that intersects with the other at (ρ-θ) and so do all the points lying on the same straight line.

Example of a linear Hough transform



2.3.2 Circle Hough transform

For this thesis we are interested in circle detection so we need to adapt our linear Hough transform in order to find circles. In a two dimensional space, a circle can be described by:

$$(x - c_x)^2 + (y - c_y)^2 = r^2 \tag{2.1}$$

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 our parameter space is two-dimensional and of course if we know nothing the parameter space is three-dimensional.

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 what was possible with the conventional Hough transform. For closer study the method of choice was the combinatorial approach 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 there is an array with a minimum value and increasing by a defined stepsize to the maximum possible radius value. For the example in this thesis the minimum is 0, the maximum radius is 1 and stepsize equal to 0.001. Introducing following scoring function $\eta(r)$ allows to find the right radius.

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

and using this in a gauss distribution

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

This is of course just the circle equation with c_x, c_y being the center of the circle, x, y are the data points and r the radius. So 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 forthe highest score can then be used to find the corresponding radius.

Runtime

The runtime of this algorithm is of $\mathcal{O}(n)$ where n is the dimension of the radius array.

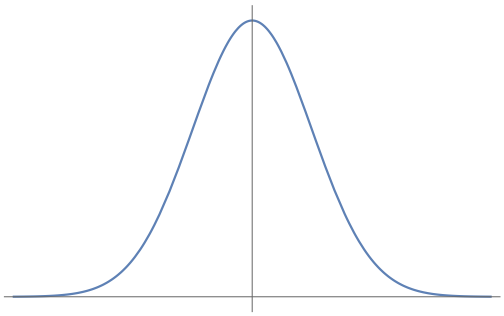


FIG. 3.1 – Using the probability density function of the normal distribution to calculate the score of a point in order to have a well defined maximum if a point lies directly on the circle and $\eta(r) = 0$.

```
DMENSION = 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

Now the radius is known and the x and y coordinate of the center (c_x, c_y) are unknown. Now instead of a one dimension the accumulator is 2 dimensional. The range of that space is simply the dimension of the plane which for this thesis is generally $[-0.5, 0.5]$. Size of the bins is 0.001. So if the plane was 1 m wide the accuracy of the accumulator is up to 1 mm. As in the one dimensional case we use the scoring function 3.1 in combination with the weight function 3.2.

Runtime

The runtime of this algorithm is $\mathcal{O}(n^2)$ where n is the dimension 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 mm $\times$ 2.5 mm.

The need (not entirely true) 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. 3.3 – 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. Weights variable is a 1001 by 1001 matrix. Again the entry with the highest score is the candidate for a possible circle center and if found stores 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 $((1,1),(1,2),...)$. So to iterate through all those grid points one would have to create 2 for-loops iterating through x and y

```
def funcB():
  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. Broadcasting now turns this one dimensional array of length n into an n by n matrix

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

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 funcA():
  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 evualted 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 funcA()
10000 loops, best of 3: 76.8 us per loop

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

So the version with broadcasting is 100 times than the double loop. And the memory consumption is moderate since they 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, that is not true. In the 2D case each grid point is a potential center for a circle only if it is not further a 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 some sub grid so only points in the surrounding sub grids.

3.1.3 Simple example of 2 circles without background

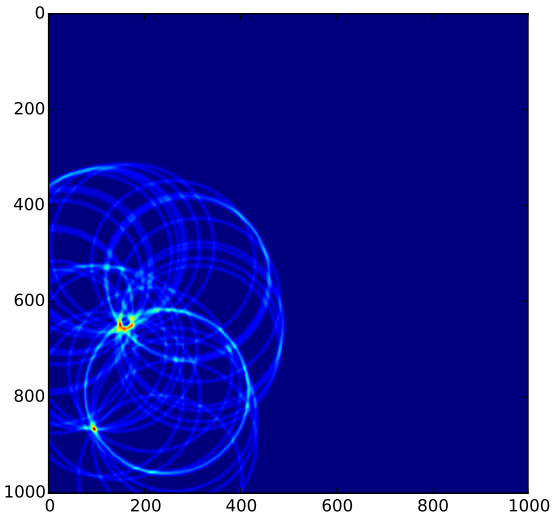


FIG. 3.4 – 2D weight matrix in the first iteration of the Hough transform algorithm. The circles have similar radiuses (0.170 and 0.158) which explains one clear maximum in the bottom left and a smeared one a bit to the top right.

3.1.4 3D: All parameters unknown

In this section all that is known are the data points and the algorithm has to retrieve both center and radius of the circles. The accumulator space is now in three dimensions. Two for the center coordinate and one for the radius. Similar as to the 2D case array broadcasting is used again to speed up the calculations of the weight. Furthermore it is the first time that 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 availabe about the circles so a condition has to be set to decide when there are no more 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])
```

Broadcasting the x, y, r arrays to speed up the calculations. To display the x, y plane for a fixed radius it is important that rbins is broadcast along the 2nd and 3rd axis.

```
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 < 2100:
    break
```

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

Since there is now no information about any of the circles it is unknown how many circles there are so a condition is needed to stop looking for circles. This algorithm uses a simple score threshold that whenever the highest score of the

weight matrix is less than a defined threshold the algorithm stops and it is assumed that all circles 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]
```

Finally with the indices found the center and radius are retrieved from the respective bins and saved to a circle dictionary. In order to avoid finding the same circle twice or use data points twice a check is being done to see whether a data point fulfills the requirement of being a circle point. If that's the case that point will be put into another list of already used points and the algorithm will not use this point anymore.

Runtime

Assuming that the radius array has the same length as the two space dimensions means that the complexity of this algorithm is of order $\mathcal{O}(N^3)$. As the 1D and 2D Hough transform the accuracy directly depends on the binning. Same as for the 2D Hough transform too match the resolution of the HPD of the RICH detector a binning of 400 makes sense if the detector dimensions are $1\,\mathrm{m} \times 1\,\mathrm{m}$.

Optimisations

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

3.2 Combinatorial approach

A circle is uniquely defined by 3 points and radius and center can be calculated. 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 and same radius for all the triplets (floating point inaccuracy not considered).

Having one background hit in addition to the 15 circle hits increases the triplet 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 used for this thesis.

1. Build all possible triples of points given the data points
2. For all the point triples calculate the center and the radius of the potential circle
3. Due to constraints in the radius many of the circles with a radius bigger than a certain threshold will be dropped.
4. Create a histogram with the radius distribution. Peaks in the radius distribution hint to a circle.
5. Scan the radius histogram for peaks and look at the center point histogram for the given radius of a peak. If there is also a peak in the center point histogram the set of the points of the triples lie on a circle with a radius and center given by the histogram peaks.

3.2.1 Generating the triples

For generating the triples 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 triple of points in a 2D plane and a, b, c the length of the sides opposite to the respective corner. The semiperimeter is defined as

$$s = \frac{a+b+c}{2} \tag{3.3}$$

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

$$R = \frac{abc}{4\sqrt{s(a+b-s)(a+c-s)(b+c-s)}} \tag{3.4}$$

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

$$\lambda_1 = a^2 \cdot (b^2 + c^2 - a^2) \tag{3.5}$$
$$\lambda_2 = b^2 \cdot (a^2 + c^2 - b^2) \tag{3.6}$$
$$\lambda_3 = c^2 \cdot (a^2 + b^2 - c^2) \tag{3.7}$$

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 noramlization) leads to the circumcenter of the triangle $\overline{ABC}$

$$\begin{pmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \boldsymbol{P'} \tag{3.8}$$

$$\frac{\boldsymbol{P'}}{\lambda_1 + \lambda_2 + \lambda_3} = \boldsymbol{P} \tag{3.9}$$

3.2.3 Drawback

There is also a drawback with this method: the combinatorics blow up with a high number of data points $\binom{N}{3}$. So for example with 200 data points (circle data and background) the number of triplets is

$$\binom{200}{3} = 1313400$$

and for 300:

$$\binom{300}{3} = 4455100$$

So the runtime of the algorithm is roughly in the order of $\mathcal{O}(N^3)$ which can be easily seen when taking the upper bound of $\binom{N}{k} \leq \frac{N^k}{k!}$ and since $k = 3$ then means $\frac{N^3}{3!}$ see figure 3.7.

Optimisation

As seen before this approach scales with $\binom{N}{3}$. This means that with 500 data points we have to create around 20 million triplets which are used for calculating radiuses and centers of circles. To qualify as a circle the algorithm needs a threshold to decide if a candidate is a circle or not. The threshold is defined as such that the minimum amount of points/circle has to be 10 in order to be considered as a circle candidate. This means that the threshold value is 120 ($\binom{10}{3} = 120$). This of course means that circles with less than 10 points will normally never be found unless another point that actually doesn't belong to

TAB. 3.1 – Example of ways to split points into 2 lists.

Pts in list 1	**Pts in list 2**
10	0
9	1
8	2
⋮	⋮
2	8
1	9
0	10

the circle lies on the circle and contributes to the radius and center histogram pushing the circle above the threshold.

But not only the amount of triplets generated is a speed bump but also the creation of this triplets takes about $\frac{N^3}{3!}$ time. So if there was a way to improve not only the amount of total triplets but also the time needed to create them it 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 the possible combinations of triplets is generated again and combined in one total list.

The problem with this approach is the possible loss of information. Since the algorithm has the threshold that defines how many entries a bin in the center histogram must have in order to be accepted as a radius/circle center. This threshold should be high enough that triplets that contain noise points do not contain to the circles but low enough that real circles with low point can still be found.

If now the data is randomly split in two lists there are only 2 ways where a circle with 10 hits will still be found namely that all the 10 points end up in the same half of the list.

So the question is what are the probabilities that splitting a data set with 10 or more points ends up with 1 list having more than 10 points. As soon as a circle has 20 points this becomes moot as always one list will have more than 10 points (see Figure 3.10).

From the figure it is clear 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.

3.2.4 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 uniformely is that they always obey a certain shape.

First we calculate the expected area of a triangle formed by three points randomly chosen from the unit square[1]. Let $A = (a_1, a_2)$, $B = (b_1, b_2)$, $C = (c_1, c_2)$ be the vertices of the random triangle T. We consider the case where $a_2 > b_2 > c_2$ which takes $\frac{1}{6}$ of the total "Volume". Fix a_2, b_2, c_2 for the moment and we can write.

$$b_2 = (1-t)a_2 + tc_2, \qquad 0 \leq t \leq 1.$$

The side AC of T intersects the horizontal level $y = b_2$ at the point $S = (s, b_2)$ with

$$s = s(a_1, c_1, t) = (1-t)a_1 + tc_1 \tag{3.10}$$

The area X of T is then given by

$$X = \frac{1}{2}|b_1 - s|(c_2 - a_2)$$

We now start integrating with respect to our six variables. The innermost integral is with respect to b_1 and gives

$$\begin{aligned} X_1 := \int_0^1 X \mathrm{d}b_1 &= \frac{1}{2}(c_2 - a_2)\left(\int_0^s (s - b_1)db_1 + \int_s^1 (b_1 - s)db_1\right) \\ &= \frac{1}{4}(c_2 - a_2)(1 + 2s + 2s^2) \end{aligned}$$

Next we integrate over b_2:

$$X_2 := \frac{1}{4}\int_0^1 \int_{a_2}^1 (c_2 - a_2)^2 \, dc_2 \, da_2 \times \int_0^1 \int_0^1 \int_0^1 (1 - 2s + 2s^2) \, dt \, dc_1 \, da_1$$

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 $a_2 < b_2 < c_2$ we multiply this result by 6 and obtain then $\frac{11}{144}$.

[1] This proof is taken from [1]

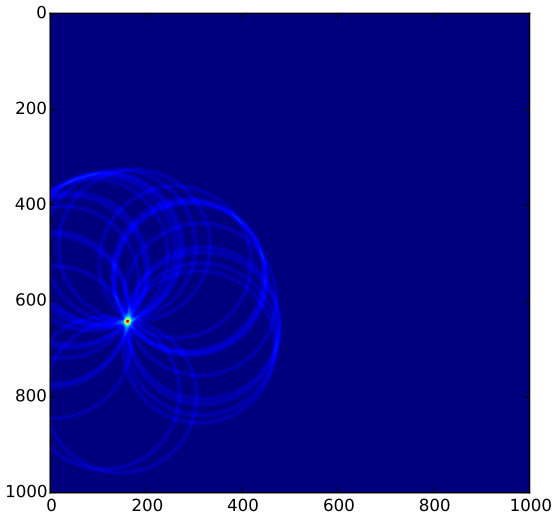


FIG. 3.5 – 2D weight matrix in the second iteration. Points that satisfied the condition being less than a certain ϵ away from the radius found in the first iteration are removed leaving (hopefully) only points available that belong to the second circles.

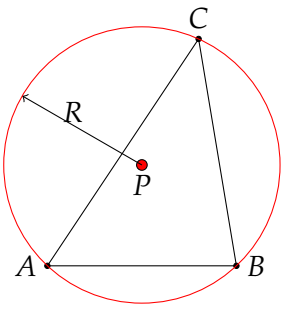


FIG. 3.6 – The circumradius (R) and the circumcenter (P) of a circle defined by three points.

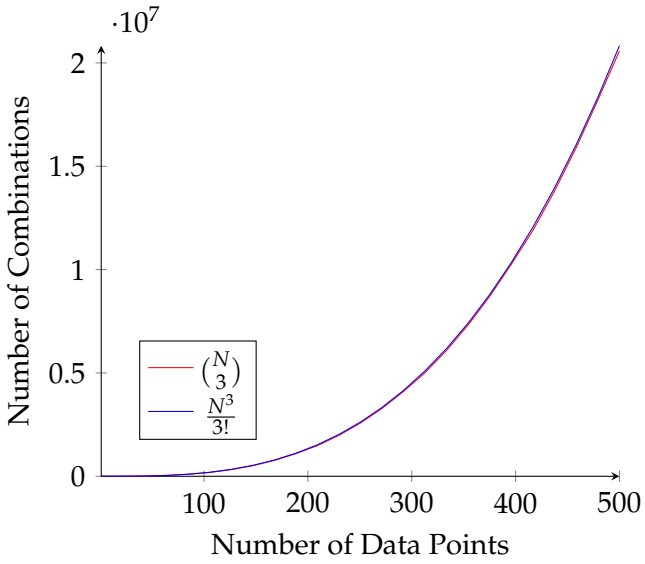


FIG. 3.7 – Scaling of the algorithm. Binomial Growth with $\binom{N}{3}$ compared with the approximation $\frac{N^3}{3!}$. So the algorithm runs in the order of $\mathcal{O}(n^3)$

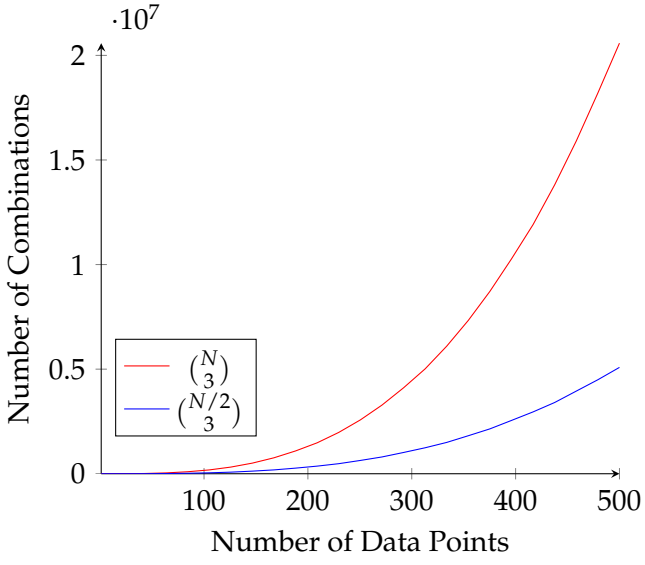


FIG. 3.8 – Number of combinations with $\binom{N}{3}$ compared to the number of combinations generated from $\binom{N/2}{3}$

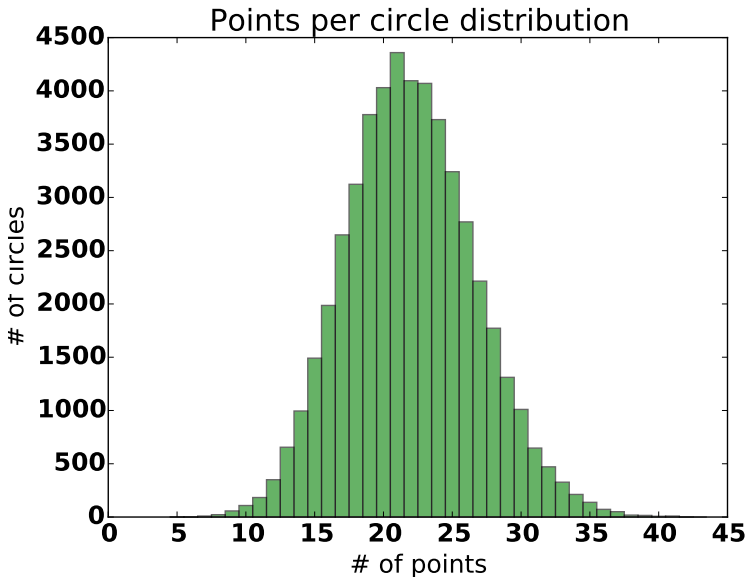


FIG. 3.9 – Distribution of points per circle.

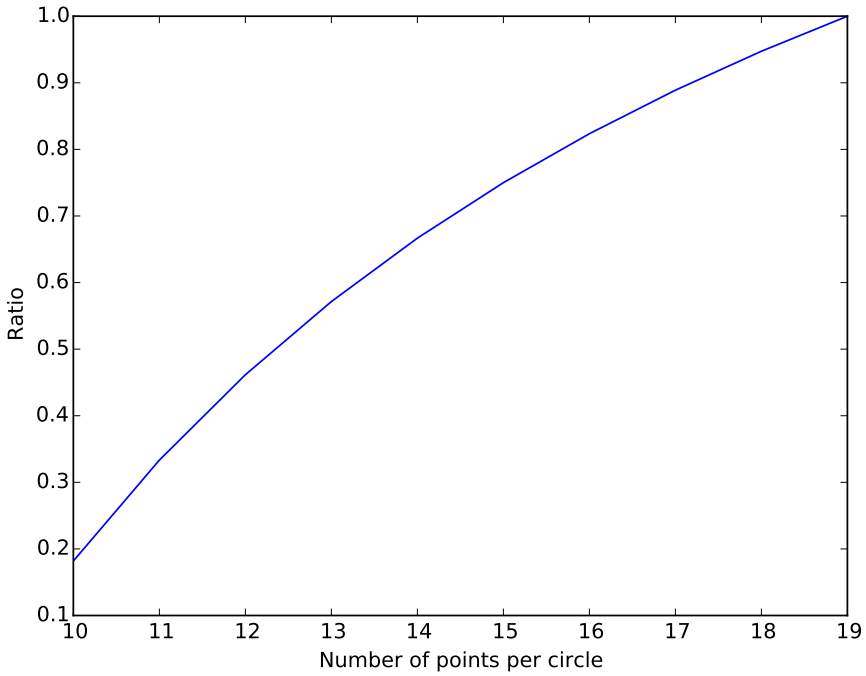


FIG. 3.10 – The probability when splitting randomly a list of x points into two that one list has more than 10 points.

4 Results

In this section results for the conventional 1D, 2D, 3D Hough transform and the combinatorial approach are presented. 1D and 2D Hough transform have no real application for the RICH detector since neither the center nor the radius is known. Nonetheless 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 and 2D Hough transform very simple data data was used. There were no physical constraints when generating the circles and their point/circle or also their radius distribution doesn't reflect the real data obtained by LHCb.

4.1 1D Hough transform results

For this section different data sets were tested.

- 1 circle and 600 background hits
- 2 circles and 0 background hits
- 5 circles and 30 background hits

For each set the radius score is shown and the final result of the algorithm. 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.

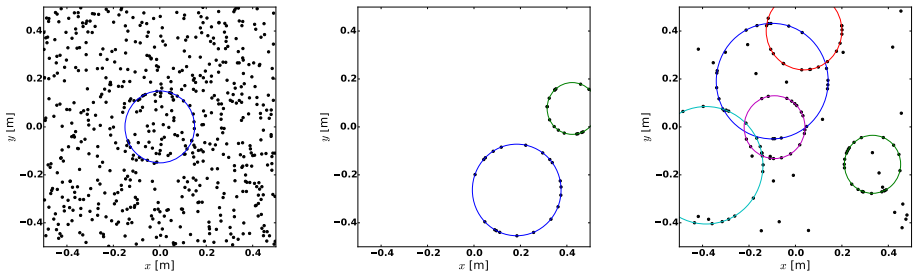
4.1.1 Overview of the results

First an overview about different test cases. The event with 1 circle and 600 background hits means to test how robust the algorithm is with a lot of background hits. The second test case means to test if the algorithm can handle two circle objects and the third event is a mix between several circles and some background hits.

There is are different plots

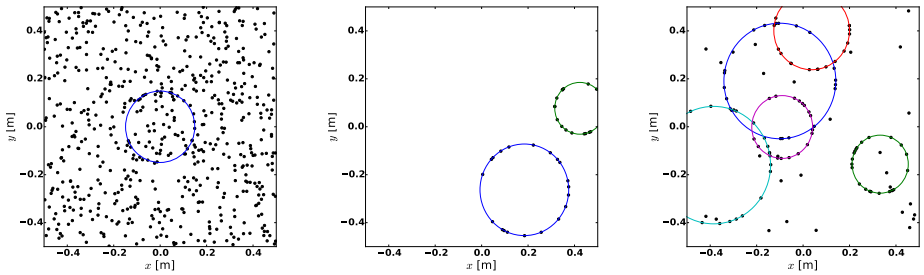
- Radius score
- Resulting circle

The radius score is a 1D plot of the score function $f(r)$. 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. There is always the same plot again with center and radius taken directly from data to compare the two results.



(A) 1 circle, 600 background hits. (B) 2 circles, 0 background hits (C) 5 circles, 30 background hits

FIG. 4.1 – Circles found by the 1D Hough transform. The circle in Figure 4.1a has its center in the origin so the algorithm did find the circle.

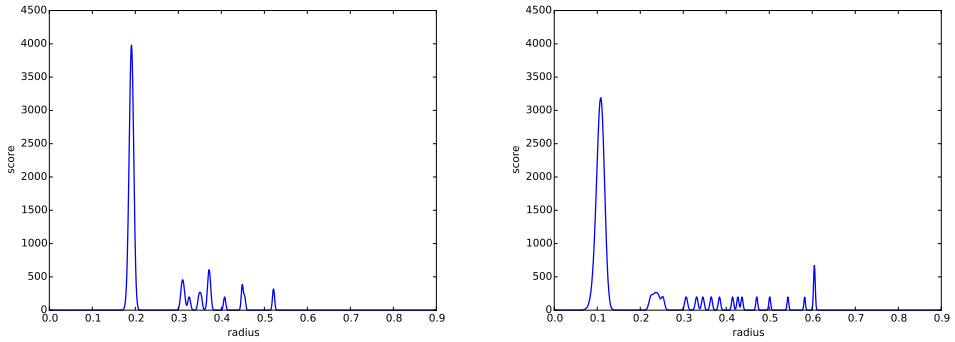


(A) 1 circle, 600 background hits. (B) 2 circles, 0 background hits (C) 5 circles, 30 background hits

FIG. 4.2 – These are the circles as generated by the simulation. All of these examples were correctly solved by the 1D Hough transform.

4.1.2 1D Hough transform - 2 circles, 0 background hits

Two circle objects have to be found in this event with zero background. Each peaks is clearly visible in their respective radius score plots. The algorithm has no problems finding two distinct circles in the plane. If two circles had the same radius and their centers are also the same (unlikely). Then the algorithm in its current state would find only one circle. A way around this could be introducing a maximum score that when a certain score is exceeded then the algorithm assumes it's more than just 1 circle. But if both circles have a low number of points this still might fail. So all in all it is not fool proof.

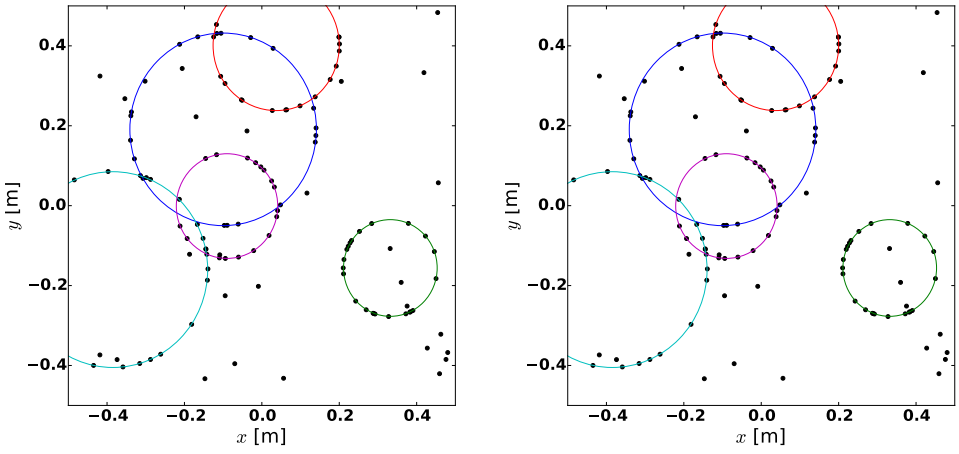


(A) 1 circle, 600 background hits. (B) 2 circles, 0 background hits

FIG. 4.3 – Radius score for the 1D Hough transform for 2 circles with 0 background. The respective radiuses are 0.191 and 0.108

4.1.3 1D Hough transform - 5 circles, 30 background hits

First we have the different radius scores for each known center. So the algorithm calculates the score for one center with all data points at a time and the radius with the highest score gets assigned to that center. Once done for all centers the algorithm is done.



(A) (B)

FIG. 4.4 – 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.

As exptected the circles can be reconstructed.

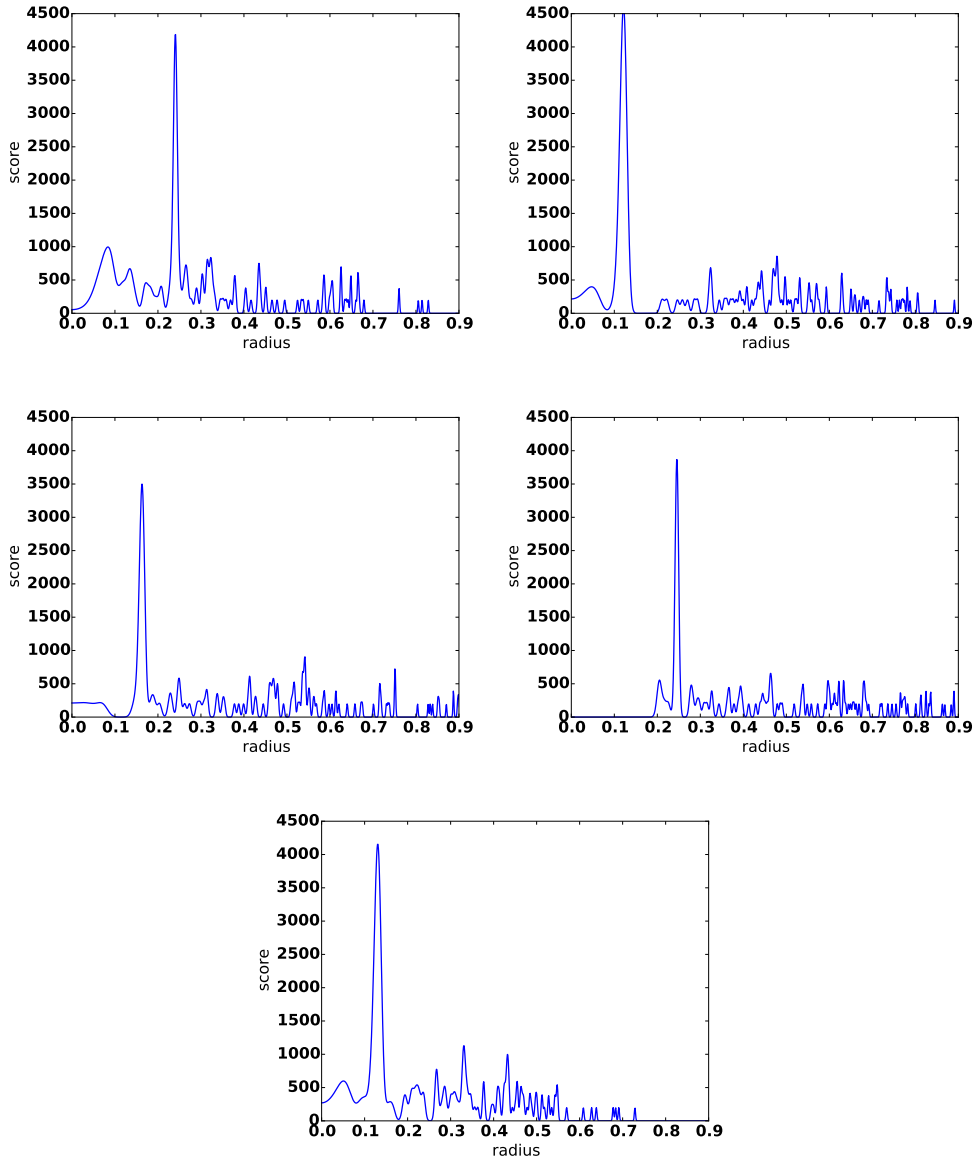


FIG. 4.5 – 1D HT: Radius scores for all the centers for an event with 5 circles. The noise contributes only very little to the radius peaks and identifying the circle candidates is easy.

4.2 2D Hough transform results

The 2D Hough transform searches in the x, y space for suitable circle centers that have a high score. The same events as for the 1D Hough transform were investigated to make a comparison about reliability. In a first overview the algorithm performs quite well and finds all the circles.

These were some of the events tackled by the algorithm (among others).

- 1 circle and 600 background hits
- 2 circles and 0 background hits
- 5 circles and 30 background hits

and additionally

- 6 circles and 200 background hits

4.2.1 Overview of the results

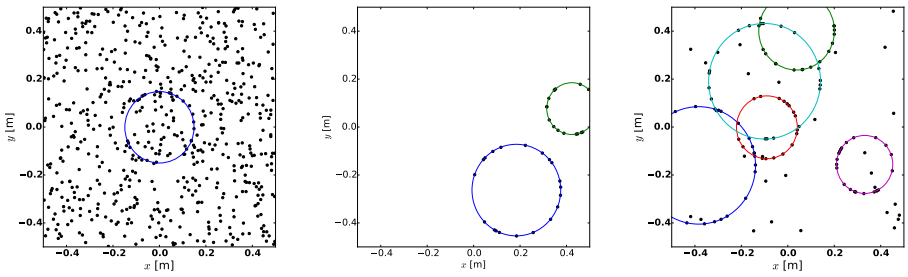
As in subsection 4.1.1 in a first step an overview of the results. The same events as before but this time tested with the 2D Hough transform. All the circles were found correctly but the example with 6 circles and 200 background hits shows that the algorithm can fail to find a circle. The reason is quite simple. Both, the yellow and magenta circle, have similar radiuses. Now 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.

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.

4.2.2 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 but as pointed out before if the two circles had a

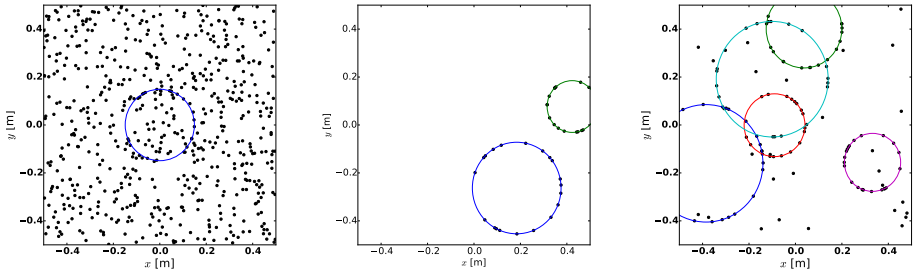


(A) 1 circle, 600 background hits. The circle is centered at 0.0/0.0.

(B) 2 circles, 0 background hits

(C) 5 circles, 30 background hits

FIG. 4.6 – Circles found by the 2D Hough transform.

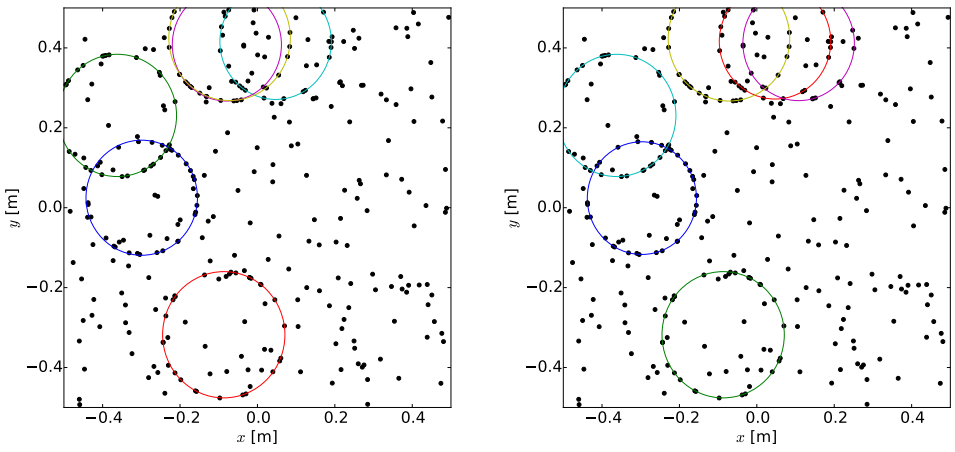


(A) 1 circle, 600 background hits.

(B) 2 circles, 0 background hits

(C) 5 circles, 30 background hits

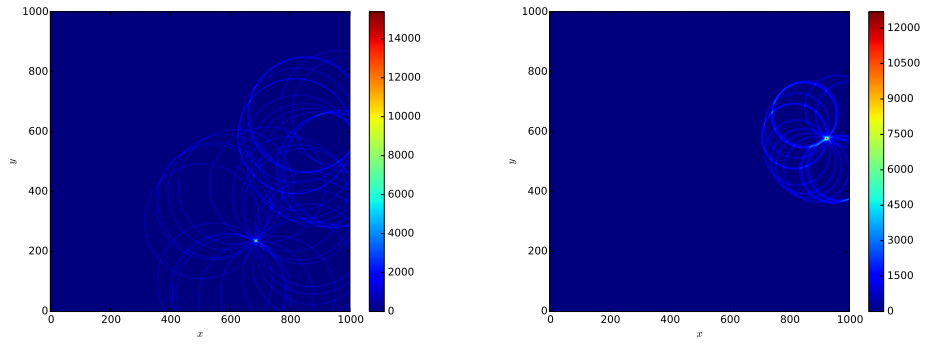
FIG. 4.7 – The circles as generated by the simulation. All of these examples were correctly solved by the 2D Hough transform.



(A)

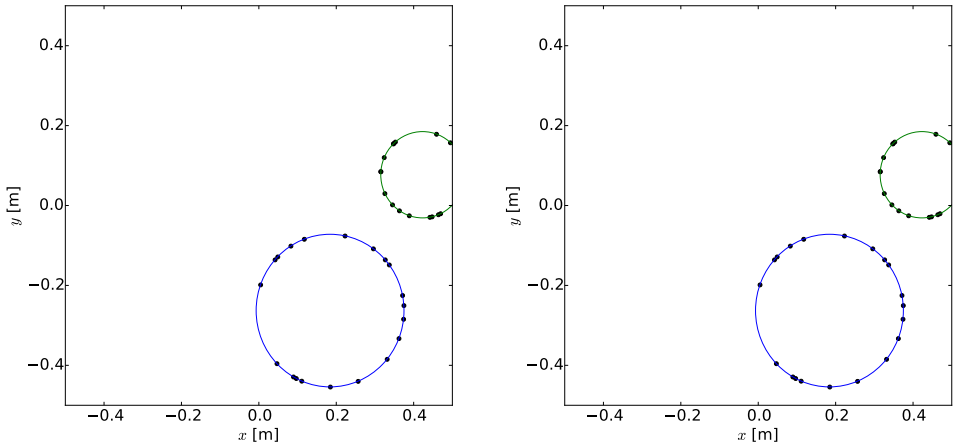
(B)

FIG. 4.8 – Circles found by the algorithm on the left and the correct results on the right. The magenta colored circle in the left image is incorrect. It replaced the yellow circle (taken from the simulated data on the right).



(A) (B)

FIG. 4.9 – Center score for the 2D Hough transform for 2 circles with 0 background.



(A) (B)

FIG. 4.10 – 2D HT: Although already shown before for completeness the 2 circle 2D Hough transform result. Left side the result from the algorithm on the right side the correct result from data.

4.2.3 2D Hough transform, 5 circles, 30 background hits

This event added more circles but also some background hits. Both were handled well by the algorithm and all the circles were found.

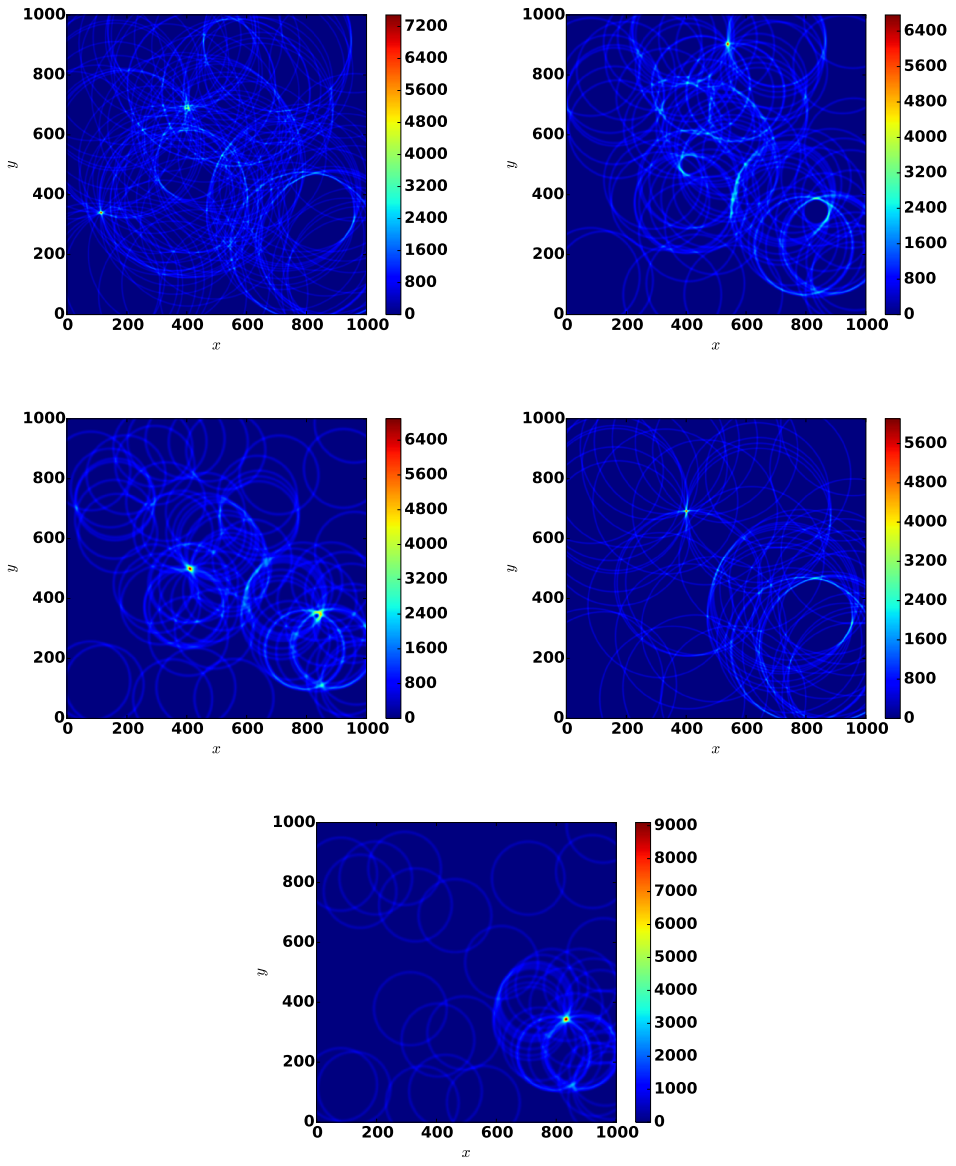


FIG. 4.11 – Center scores for all the centers for an event with 5 circles.

4.2.4 2D Hough transform, 6 circles, 200 background hits

As briefly discussed in the overview the algorithm does make mistakes as seen in this next event. 6 circles were generated with 200 background hits. The interesting part is not that amount of circles or the amount of background hits

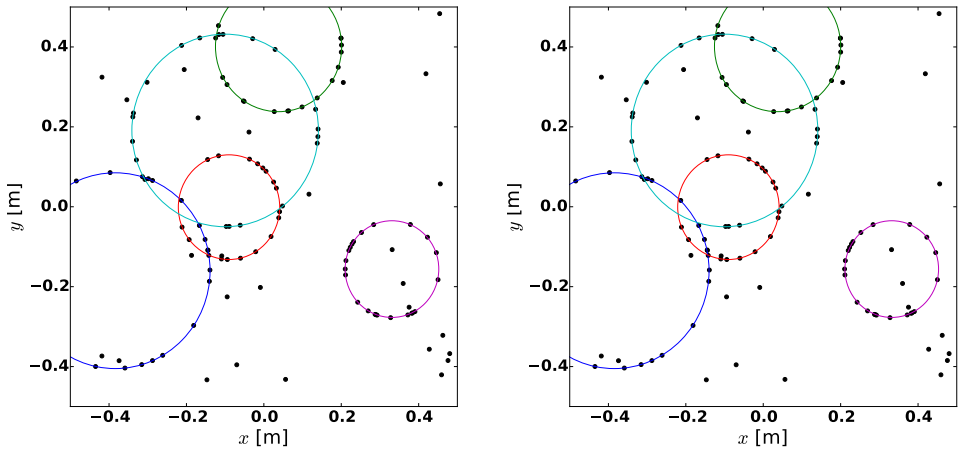


FIG. 4.12 – The reconstructed result on the left while the circles from the simulated data is on the right.

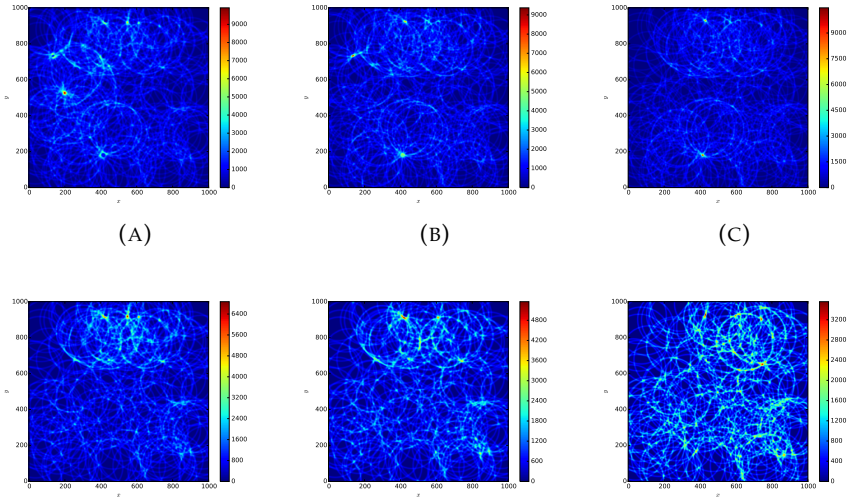
posed a problem but the properties of the circles (center coordinate and radius). In this example there is a misidentification of a circle with points of another.

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

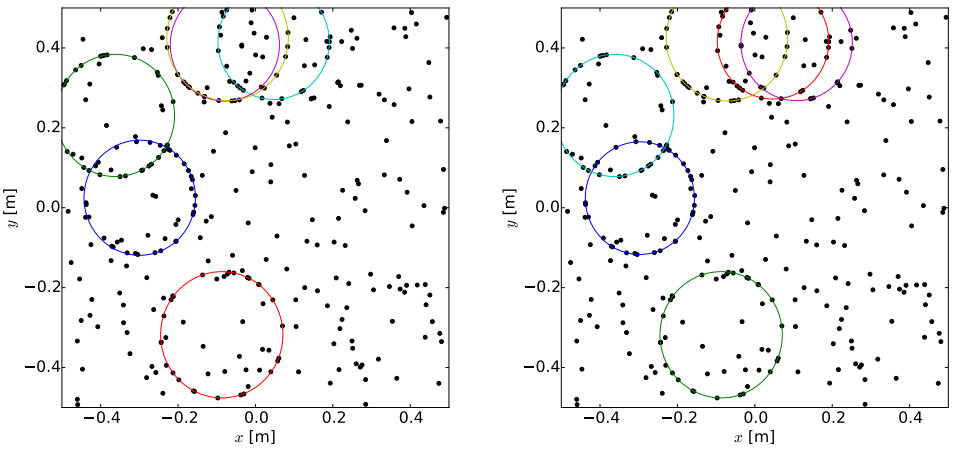
In the example before σ was equal to 0.001 while the space dimension was 1. So if the detector was 1 m per dimension a hit 1 mm off the perfect location contributes still more half of the score off the perfect location. If we set $\sigma = 0.0005$ a point 0.001 away from the perfect location it only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



(A) (B) (C)

(D) (E) (F)

FIG. 4.13 – 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.



(A) (B)

FIG. 4.14 – On the left side the wrong result obtained by the 2D Hough transform and the correct one on the right side

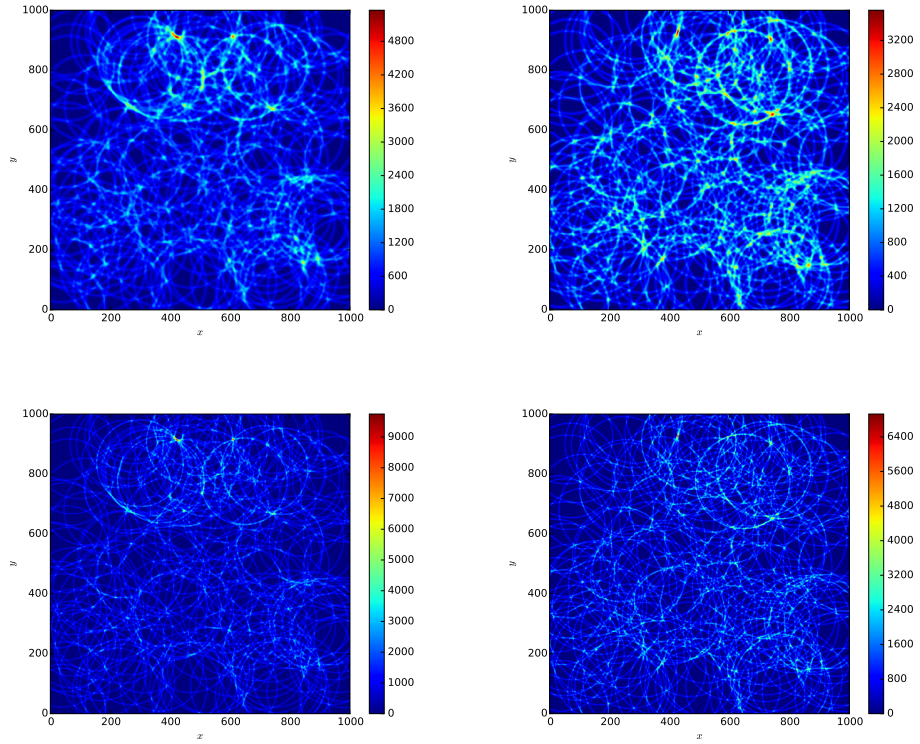
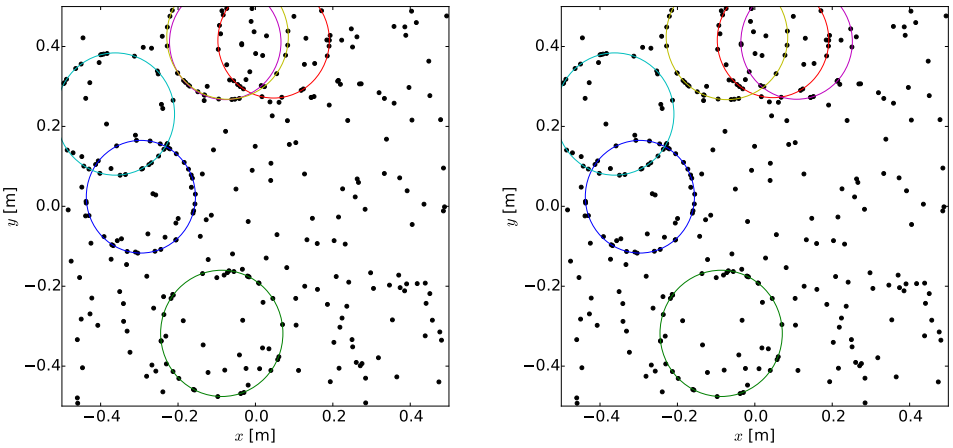


FIG. 4.15 – Old center score in the top row for circle 5 (magenta) and 6 (yellow). And below the same circles this time with the new $\sigma = 0.0005$. It is well visible how big the influence of sigma is for the center score. With a σ of 0.001 just by eye there seem to be many similar maxima whereas with a $\sigma = 0.0005$ the maximas get much more distinct and the maximum score is also becomming bigger. Where the high score in the top row is ≈ 5000 and 3500 in the bottom it is ≈ 9500 and 6500.



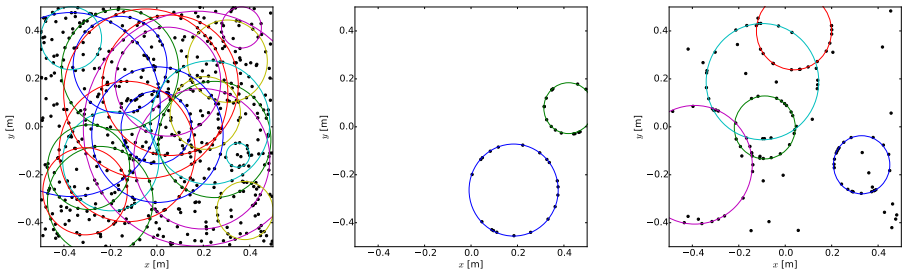
(A) (B)

FIG. 4.16 – 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. 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.

- 1 circle and 600 background hits
- 2 circles and 0 background hits
- 5 circles and 20 background hits
- 6 circles and 200 background hits

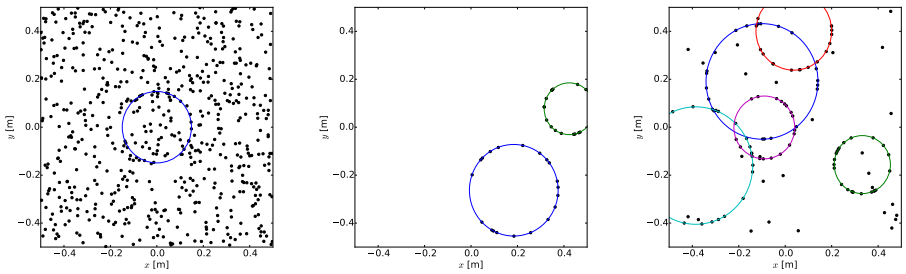


(A) 1 circle, 600 background hits. The circle is centered at 0.0/0.0.

(B) 2 circles, 0 background hits

(C) 5 circles, 30 background hits

FIG. 4.17 – Circles found by the 3D Hough transform. Figure 4.17a shows that with a low enough signla/noise ratio even this seemingly simple event fails to be caluclated correctly. Even though the algorithm found the real circle it also found 22 others.

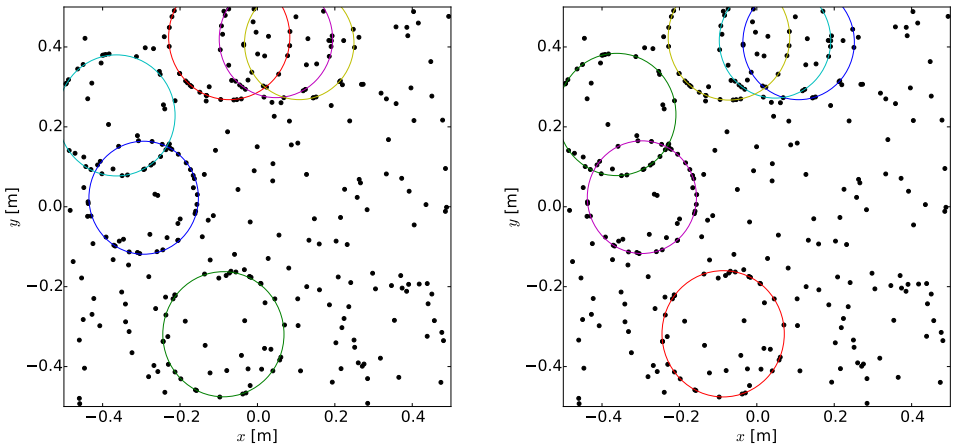


(A) 1 circle, 600 background hits.

(B) 2 circles, 0 background hits

(C) 5 circles, 30 background hits

FIG. 4.18 – These are the circles as generated by the simulation. The 1 circle 600 background event was problematic for the 3D Hough transform. Tuning the score Threshold could improve it for this particular case. The events with little or no background were solved correctly.



(A) (B)

FIG. 4.19 – Circles found by the algorithm on the left and the correct results on the right. Where the 2D HT failed the 3D Hough transform solves it correctly.

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 most 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. A way to ammend this particular case would be tuning the score threshold. The more background points there are the higher gets the 'background score' meaning score generated by only background hits. 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).

Tuning the score parameter

As mentioned before one way to tune the 3D Hough transform is changing the score threshold. The problem with the example with 600 background is that too many background hits increase the overall score of the whole score matrix and fake circles above this threshold make it into the result.

The scores for each of the circles in this example are the following:

TAB. 4.1 – All the scores for the 1 circle 600 background hits circles. Threshold score was 3500

8122	6256	5672	5654
5373	5290	5047	5000
4727	4647	4533	4530
4370	4308	4234	4159
4136	4127	4104	3829
3702	3637	3593	

4.4 Combinatorial approach results

This method was tested against a different set of events. 10'000 different events generated in an attempt to simulate LHCb data. First some analysis on the data.

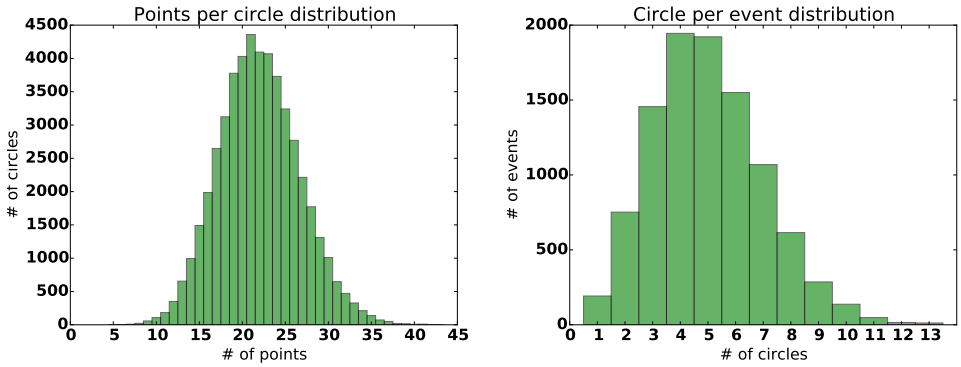


FIG. 4.20 – Points per circle distribution on the left, circle per event distribution on the left

The the main parameters for this algorithm are

- Histogram thresholds (radius and center)
- Distance between duplicate rings
- Difference between duplicate ring radiuses
- Distance between a calculated ring and the real ring
- Difference between calculated radius and real radius

Getting rid of duplicates

Without cleaning up the algorithm finds a lot of duplicates. Imagine the 1D radius histogram where two almost adjacent bins have a high score as seen in

Figure 4.21. They wont 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 a independent circles and without cleaning for duplicates the result can be seen in Figure 4.22

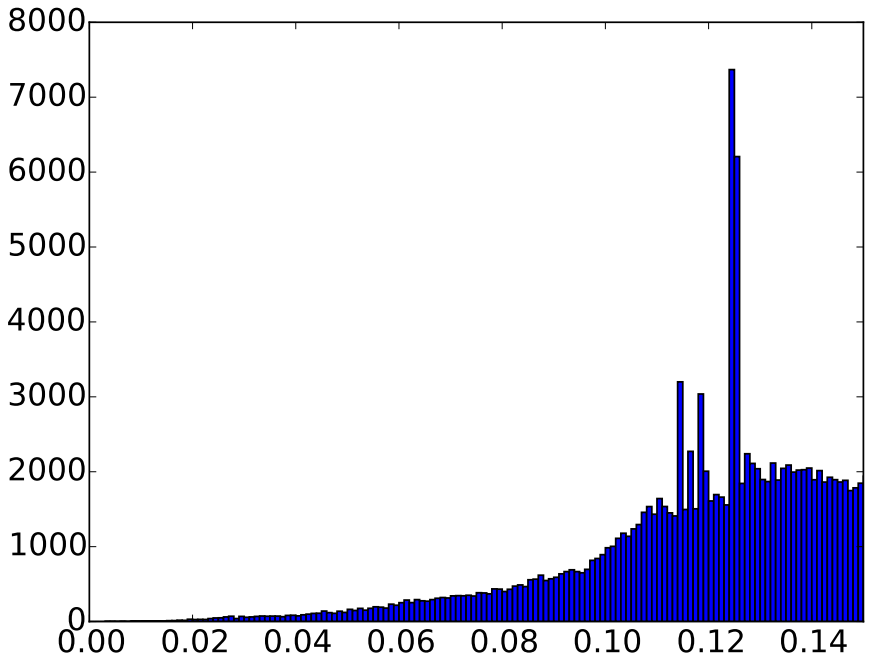


FIG. 4.21 – Two adjacent bins with a high score

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
```

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 the basic algorithm is done and in Figure 4.22 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
```

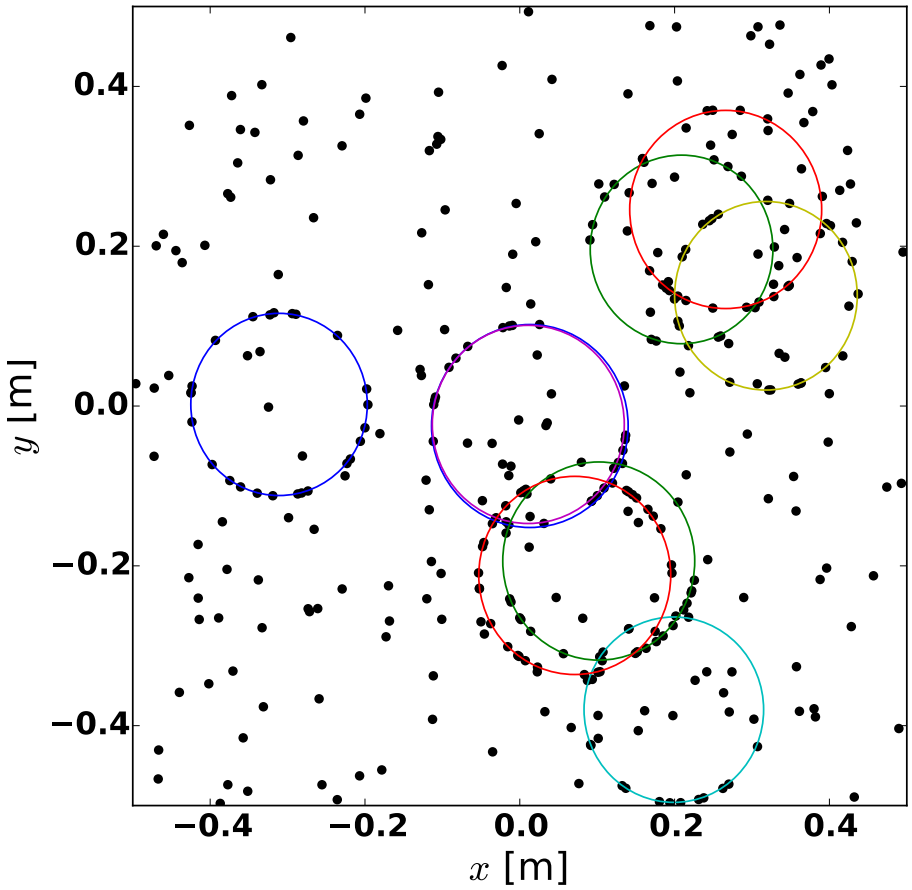


FIG. 4.22 – The result of the combinatorial approach before cleaning up the duplicates. Blue and purple are basically the same circle.

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 developped for the circle detection.

Bibliography

[1] 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.

[2] LHCb Collaboration. *The LHCb Detector*. Online. URL: http://lhcb-public.web.cern.ch/lhcb-public/en/Detector/Detector-en.html.

[3] 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.

[4] RICH LHCb. "Technical Design Report". In: *CERN/LHCC* 37 (2000), p. 2000.

[5] A Powell. *Particle identification at LHCb*. Tech. rep. 2011.