Breaking the speed limit with multimode fast scanning of DNA by Endonuclease V (Supplementary code)

Arash Ahmadi et al.

Contents

1 Data acquisition

The details of data acquisition are explained in the method section of the paper and here a brief overview of the data is given, in addition to some experimental conditions in which data has been acquired.

1.1 Experimental details

The red laser line with wavelength of 647nm was used to excite the ATTO 647N dye.

The illumination laser power in the sample plane were measured to be $0.05 - 1kW/cm^2$.

The exposure time was 7.5 ms and 23.5 ms for EndoV and hOGG1, respectively. The optimum exposure time for detecting maximum possible signal intensity in maximum lifetime of the fluorescent dyes was decided empirically and after several initial observations of protein-DNA interactions.

The power of the trapping laser in the sample plane was measured to be $1.3 - 18mW$.

1.2 Structure of data

The raw data sets are streams of videos with a length ranging from 20 k frames to 200 k frames. Depending on the sample condition, each data set can contain varying numbers of protein-DNA interactions recorded as trajectories of protein scanning along the DNA.

Total amount of Data:

- Human OGG1 (hOGG1): 71 data sets, 13.7 GB
- Wild-type EndoV (wt-EndoV): 127 data sets, 72.2 GB
- Wedge mutant EndoV (wm-EndoV): 67 data sets, 40.5 GB

The interaction rate in these data sets is very low, since the concentration of proteins in the interaction chamber were kept low. Therefore the first step is to detect these trajectories.

2 Localization of the signal from the video files

In detecting the protein-DNA interaction as trajectories we had two main issues with available tracking plugins in FIJI:

1) The efficiency of the analysis was drastically lowered when attempting to process very large data sets and it could take unreasonably long time to process the data.

2) The existing plugins were not fully automated and needed a lot of user intervention which could introduce user bias in the analysis. In addition, the need for user intervention further lowered the efficiency of the analysis since trajectories had to be detected among a very large number of frames.

Therefore we decided to develop a code in R to perform the tracking and further analysis in a more efficient and automated way.

2.1 Localization of the single molecule signals

The signal from single molecules were localized in *FIJI* [1] using *ThunderSTORM* [2], and the camera setup configuration in the software was set as:

- pixel size : 112nm
- Photoelecrons per A/D count : 20.02
- Base level A/D count : 991.34
- Em gain: 398.01

The script for automatic localization of tiff files performed in FIJI:

```
dir1 = getDirectory("");
dir2 = getDirectory("");
dir3 = getDirectory("");
list = getFileList(dir1);
setBatchMode(true);
for (i=0; i<list.length; i++) {
    showProgress(i+1, list.length);
    open(dir1+list[i]);
suf1= ".json";
suf2=".png";

run("Run analysis", "filter=[Wavelet filter (B-Spline)] scale=2.0 order=3
    detector=[Local maximum] connectivity=8-neighbourhood threshold=std(Wave.F1)
    estimator=[PSF: Integrated Gaussian] sigma=1.6 method=[Weighted Least squares]
    full_image_fitting=false fitradius=3 mfaenabled=false
    renderer=[Averaged shifted histograms] magnification=5.0 colorizez=false
    shifts=2 repaint=50 threed=false");

run("Export results", "filepath=["+dir2+list[i]+suf1+"] fileformat=JSON id=true
    frame=true sigma=true chi2=true bkgstd=true intensity=true saveprotocol=true
    offset=true uncertainty=true y=true x=true");

run("Close");
saveAs("tiff", dir3+list[i]);

run("Close");
run("Close");
}
```

All signals detected in each data set are projected into single images.

Fig.1 is an example of projected signals localized in one data set :

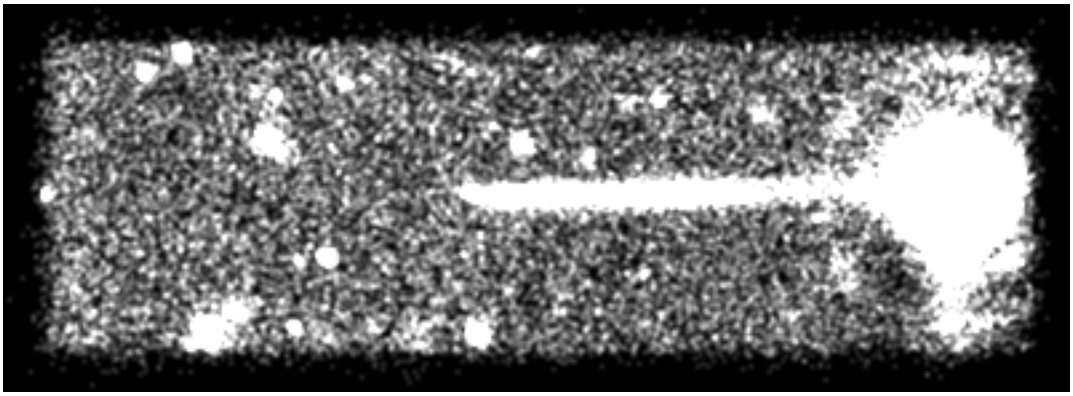

Figure 1: Localization picture of one data set

2.2 Packages used in R

From this point the R codes start and here is a list of R packages that will be used:

```
library(magrittr)
library(tidyverse)
library(viridis)
library(broom)
library(plotly)
library(e1071)
library(party)
library(data.table)
library(microbenchmark)
library(dplyr)
library(shiny)
library(stringr)
library(scales)
library(Rcpp)
library(RcppRoll)
library(zoo)
library(XML)
library(ggplot2)
library(jsonlite)
library(caret)
library(R.matlab)
library(svglite)
library(markovchain)
library(igraph)
```

2.3 Localization of the beads

For each data set beads were localized using *TrackMate* [3] plugin in FIJI and the output was used for localization of DNA in tracking code in R. Here are the scripts used for this purpose in FIJI and R.

```
dir1 = getDirectory("");
list = getFileList(dir1);
setBatchMode(true);
for (i=0; i<list.length; i++) {
```

```
    showProgress(i+1, list.length);
    open(dir1+list[i]);
    Name = list[i];
    makeRectangle(3, 19, 9, 7);
    run("Plot Z-axis Profile");
    saveAs("Results", "xml\\"+ list[i]+".txt");
    run("Clear Results");
    while (nImages>0) {
          selectImage(nImages);
          close();
      }
     print(i);
      }
```

```
rm(list = ls())

# get the list of xml outputs from FIJI

files <- list.files(pattern = "\\.xml$")
Output1 = 0
Output = 0
pl = 0

# read the average intensity from the text files to define
# the timepionts of bright field on-off state read the xml
# file and define the bead position during the brigh field on
# state

for (j in 1:length(files)) {
   Intensity <- read.table(file = paste(strtrim(files[j], (nchar(files[j]) -
      4)), ".txt", sep = ""), header = T, strip.white = T)
   IntData <- data.frame(seq(1:nrow(Intensity)), Intensity$Mean)
   names(IntData) <- c("FN", "MI")
   print(j)
   data <- xmlParse(files[j])
   Fpath <- "//Spot/@FRAME"
   Xpath <- "//Spot/@POSITION_X"
   Ypath <- "//Spot/@POSITION_Y"

   DF <- data.frame(FN = as.integer(sapply(data[Fpath], as,
      "integer")) + 1, XP = as.numeric(sapply(data[Xpath],
      as, "numeric")), YP = as.numeric(sapply(data[Ypath],
      as, "numeric")))
   DF = dplyr::left_join(DF, IntData, by = "FN")
   if (length(DF$MI[DF$MI > 3000]) < 50) {
      DF$MI <- DF$MI * 2
   }


   DF <- (DF %>% filter(XP > 50, YP > 8, YP < 22, MI > 3000))

   DF <- DF %>% filter(XP > (mean(XP) - 7), YP > (mean(YP) -
      7), YP < (mean(YP) + 7))
```

```
row = nrow(Intensity)
k = 0
p = 0
FrameNumber = 0
XPosition = 0
YPosition = 0
Status = ""
a <- 50
for (i in 1:row) {
   l <- length(DF$FN[DF$FN == i])
   if (l == 1) {
      FrameNumber[i] <- i
      XPosition[i] <- DF$XP[i - k + p]
      YPosition[i] <- DF$YP[i - k + p]
      Status[i] <- "Detected"
   } else if (l == 0) {
      if (i <= a) {
         FrameNumber[i] <- i
         XPosition[i] <- mean(DF$XP[(i - k + p + 1):(i -
           k + p + a)])
         YPosition[i] <- mean(DF$YP[(i - k + p + 1):(i -
           k + p + a)])
         k <- k + 1
         Status[i] <- "Forward estimated"
      } else {
         FrameNumber[i] <- i
         XPosition[i] <- mean(XPosition[(i - a):(i - 1)])
         YPosition[i] <- mean(YPosition[(i - a):(i - 1)])
         k <- k + 1
         Status[i] <- "Backward estimated"
      }
   } else if (l > 1) {
      if (i <= a) {
         FrameNumber[i] <- i
         XPosition[i] <- mean(DF$XP[(i - k + p + 1):(i -
           k + p + a)])
         YPosition[i] <- mean(DF$YP[(i - k + p + 1):(i -
           k + p + a)])
         p <- p + l - 1
         Status[i] <- "Forward estimated"
      } else {
         FrameNumber[i] <- i
         XPosition[i] <- mean(XPosition[(i - a):(i - 1)])
         YPosition[i] <- mean(YPosition[(i - a):(i - 1)])
         p <- p + l - 1
         Status[i] <- "Backward estimated"
      }
   }
}
MeanItn <- Intensity$Mean

DF1 <- data.frame(FrameNumber, XPosition, YPosition, Status,
   MeanItn)
```

```
   FileName <- rep(files[j], row)

   Output1 <- data.frame(FileName, DF1)
   Output <- rbind(Output, Output1)
}
Output$XPosition <- Output$XPosition * 112
Output$YPosition <- Output$YPosition * 112


saveRDS(Output, "~/data/mEndoV/outputs/mEndoVBead.rds")
```

3 Tracking

The localized signals from each data set are treated individually through a tracking code. The code has two main scripts:

1. the interface in which user chooses the targeted data set and runs the code part by part.
2. the function source that provides the interface with operating functions.

3.1 The user interface code

In this part the user is choosing the data sets one by one and sequentially performing:

1) spatial filtering (Fig.2 and Fig.3).
2) trajectory detection (Fig.4).
3) noise exclusion; only trajectories that last longer than 5 frames and the proteins have moved at least 300 nm will pass this step (Fig.5).
4) visual inspection; here the detected trajectories are visually inspected.

```
##### directory and funcitons

Output <- readRDS("EndoV.rds")
Info <- read.table("EndoV.txt",
   header = T)

rm(list = ls()[!(ls() %in% c("Output", "Info"))])

JsonFilePath <- "mEndoV_10mMNaCl_2016102906_7.5mspf_112nmpp_1-10_2000X.tif.json"
NameInfo <- "mEndoV_10mMNaCl_2016102906_7.5mspf_112nmpp_1-10_2000X"

source("~/R/codes/Tracking/function.R")

##### Part 1: spatial filtering

# initial visualization
SourceData <- fromJSON(JsonFilePath)
RawDataMat <- MakeItMatrix(RawData = SourceData, InputData = SourceData)
RawDataMatNA <- MakeZeroNA(RawDataMat)
LongRaw <- MakeLongForm(RawDataMatNA)
```

```
ss <- PlotXY1FrameColor(LongRaw)

# after seeing the figures from part 2 the used sets the
# approximate boundries of where the DNA lays based on the
# traces of the scanning proteins

temp1 <- FilterIt(RawData = SourceData, xmin = 2500, xmax = 7400,
   ymin = 1200, ymax = 1900, intenmax = 1000, intenmin = 15)
FilteredData <- temp1[[1]]
FilInfo <- temp1[[2]]
FilDataMat <- MakeItMatrix(RawData = SourceData, InputData = FilteredData)
FilDataMatNA <- MakeZeroNA(FilDataMat)
LongFil <- MakeLongForm(FilDataMatNA)

ss <- PlotXY2(LongFil, LongRaw)
ss

##### Part 2: trajectory detection

# here the code automatically detect the trajectories, this
# step can take up to several minutes

# dxmax is the max of traveling distance within each frame
# and dymax is the max of DNA flactuation in y direction this
# value was chosen imperically based on investigation of
# several manually detected trajectories.

temp2 <- FindTrajectory(FilteredDataSet = FilDataMat, dxmax = 600,
   dxmin = 0.1, dymax = 200)
DetDataMat <- (temp2[[1]])
DetInfo <- temp2[[2]]
DetDataMatNA <- MakeZeroNA(DetDataMat)
LongDet <- MakeLongForm(DetDataMatNA)

ss <- PlotXY3(LongDet, LongFil, LongRaw)
ss

##### Part 3: Noise exlusion

# here trajectories are filtered based on min of displacement
# and frame numbers. the bourdaries are set at 300 nm and 5
# frames.

# one output of this step is the information of registered
# trajectories with which the actuall videos are cut and
# trajectories can be visually inspected.

temp3 <- ExtractTrajectory(DetectedLongForm = LongDet, xRange = 300,
   FrameLength = 5)
ExtDetTra <- temp3[[1]]
ToMacro <- temp3[[2]]
TrajInfo <- temp3[[3]]
```

```
write.table(ToMacro, paste("FrameCuts/",
   NameInfo, ".txt", sep = ""), sep = "\t", col.names = T)

##### Part 4: visual inspection

# before this step the script in FIJI should be ran and the
# acutall videos should be cut, in the video the trajectories
# are labled with a number and based on those numbers they
# can be visually inspected. The trajectories in which the
# protein clearly moves on DNA are selected here and later
# will be used to localize the DNA

ggplotly(PlotFX1TrajOnFra(ExtDetTra))
ggplotly(PlotFX1TrajOnSeq(ExtDetTra))

ExtDetTra <- ExtDetTra %>% mutate(v = r)
ggplotly(PlotFX1YesTrajOnFrame(ExtDetTra))
ggplotly(PlotFX1YesTraj(ExtDetTra))



ggplotly(PlotFX1SingleTraj(ExtDetTra, LongFil, list(1)))
# see the individual trajectory and inspect it through the video

a <- c()
q <- "Yes"
while (q == "Yes") {
   a <- c(a, readinteger())
}

ExtDetTra$v[!is.na(ExtDetTra$s) & ExtDetTra$v == "Yes"] <- "No"

for (i in 1:length(a)) {
   ExtDetTra$v[!is.na(ExtDetTra$s) & ExtDetTra$s == a[i]] <- "Yes"
   # if accepted skip this line, if not input trajectory number
}

ggplotly(PlotFX1YesTraj(ExtDetTra))

n <- rep(NameInfo, nrow(ExtDetTra))

Output1 <- data.frame(ExtDetTra, n)
# watch out for the first dataset of each protein
Output <- rbind(Output, Output1)

Info1 <- data.frame(NameInfo, FilInfo, DetInfo, TrajInfo)
Info <- rbind(Info, Info1)

saveRDS(Output, "OutPut/EndoV.rds")
write.table(Info, "OutPut/EndoV.txt",
   sep = "\t", col.names = T)
```

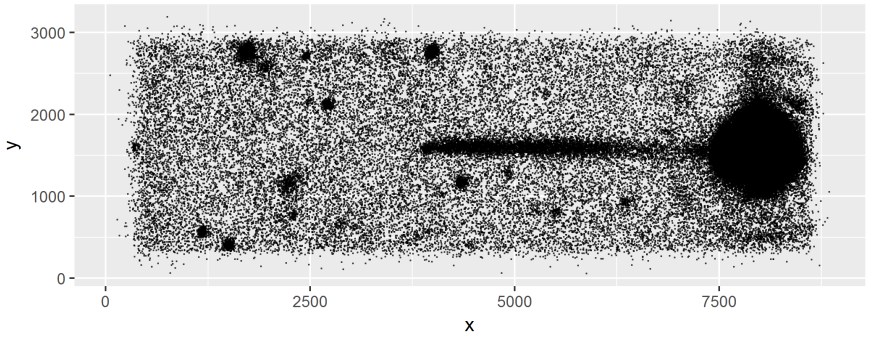

Figure 2: output of part 1

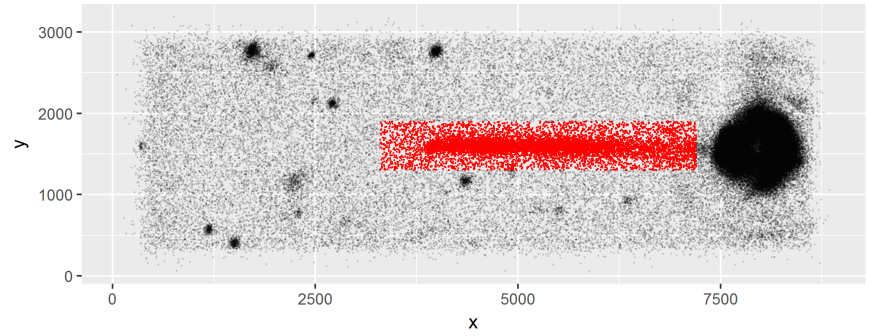

Figure 3: second output of part 1

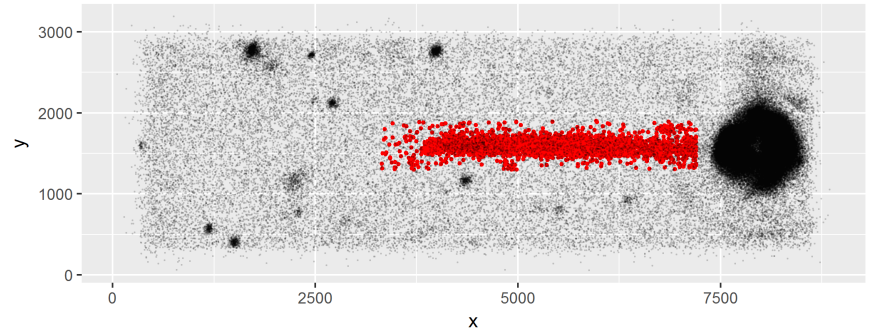

Figure 4: output of part 2

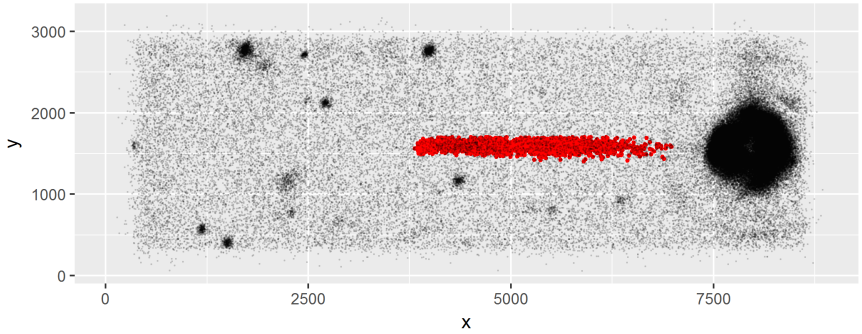

Figure 5: output of part 3

3.2 Operating function

Here is the list of functions that are used in the interface.

```
#####
# construction of the matrix from outputs of thunderSTORM

MakeItMatrix <- function(RawData,InputData,DataMat)
{
# creation of a matrix from rawdata, this matrix can be used
# for visualization of the raw data parallel to the filtered
# and detected data
row = max(RawData$frame) # max number of frame
SignalPerFrame <- 0
for(i in 1:row)
{  # this loop is for extractoin of number of signals in one
# frame which later will be the row numbers
  SignalPerFrame[i] <- length(InputData$frame[InputData$frame==i])
}


col = max(SignalPerFrame) # max number of signal per frame
DataMatx <- matrix(0,row,col)
DataMaty <- matrix(0,row,col)
DataMatf <- matrix(0,row,1)
k = 0
p = 0

for(i in 1:row)
{
  DataMatf[i,1] <- i

  l <- length(InputData$frame[InputData$frame==i])
  if (l==0)
  {
    DataMatx[i,] <- 0
    DataMaty[i,] <- 0
    k <- k+1
  }
  else
```

```
  {
    for(j in 1:l)
    {
      DataMatx[i,j] <- InputData$`x [nm]`[i-k+p+j-1]
      DataMaty[i,j] <- InputData$`y [nm]`[i-k+p+j-1]

    }
    p <- p+l-1
  }
}
DataMat <- data.frame(DataMatx, DataMaty)
for (i in 1:col)
  {
    names(DataMat)[i]<-paste("x",i, sep = "")
    names(DataMat)[col+i]<-paste("y",i, sep = "")
  }
return(DataMat)
}

#####
# For trajectory detection loop it's better to have zero
#value when there is a gap,

MakeZeroNA <- function(InputDataSet)
{
  InputDataSet[InputDataSet==0] <- NA
  return(InputDataSet)
}

#####
# Detecting the trajectories based on user input for max
#traveled distance per frame

FindTrajectory <- function(FilteredDataSet, dxmax, dxmin, dymax)
{

  info <- 0
  row <- nrow(FilteredDataSet)
  col <- ncol(FilteredDataSet)/2
  # detection loop
  for(i in 1:(row-1))
  {
    for(j in 1:(col))
    {
      if (FilteredDataSet[i,j]!=0)
      {
        if(j!=col)
        {
          if (sqrt((FilteredDataSet[i,j])^2 +
             (FilteredDataSet[i,(j+col)])^2) -
             sqrt((FilteredDataSet[i,(j+1)])^2 +
             (FilteredDataSet[i,(j+col+1)])^2) < 200 &
             abs(FilteredDataSet[i,j]- FilteredDataSet[i,(j+1)]) <200)
```

```
          # if two signal has been detected within one PSF in one frame,
          #consider the mean of the two
      {
        FilteredDataSet[i,j] <- (FilteredDataSet[i,j]+
                                 FilteredDataSet[i,(j+1)])/2
        FilteredDataSet[i,j+col] <- (FilteredDataSet[i,j+col]+
                                 FilteredDataSet[i,(j+col+1)])/2
        FilteredDataSet[i,j+1] <- 0
        FilteredDataSet[i,j+col+1] <- 0
      }
    }
  deltax<- abs(FilteredDataSet[i,j]-FilteredDataSet[i+1,1:col])
  deltay<- abs(FilteredDataSet[i,(j+col)]-
                 FilteredDataSet[i+1,(col+1):(2*col)])
  min <- which.min(deltax)
  # guarantees that we follow the closest signali in the next frame

  if(deltax[min]<dxmax & deltax[min]>dxmin & deltay[min]<dymax)
  {
    if (i!=1)
    {
      if (FilteredDataSet[i-1,j]!=0)
      {# this is for separation of two consacutive trajectories
        if (abs(FilteredDataSet[i,j]-FilteredDataSet[i-1,j])> dxmax |
            abs(FilteredDataSet[i,j]-FilteredDataSet[i-1,j])< dxmin |
            abs(FilteredDataSet[i,(j+col)]-
            FilteredDataSet[i-1,(j+col)])> dymax)
         { # basically, if this is the first point
          #then make the previous point zero, make sure it is
          #not following another trajectory

          print("first point detected")
          print(i)
          print(j)

          FilteredDataSet[i-1,j] <- 0
          FilteredDataSet[i-1,(j+col)] <- 0
         }
      }
   }
  if (abs(FilteredDataSet[i+1,min]-FilteredDataSet[i,j]) ==
      min(abs(FilteredDataSet[i,1:col]-FilteredDataSet[i+1,min])))
  { # in case there are two signal within the dxmax and dymax,
    #this help to choose the clsoest signal to the signal in the last frame
    ax <- FilteredDataSet[i+1,j]
    FilteredDataSet[i+1,j] <- FilteredDataSet[i+1,min]
    FilteredDataSet[i+1,min] <- ax
    ay <- FilteredDataSet[i+1,(j+col)]
    FilteredDataSet[i+1,(j+col)] <- FilteredDataSet[i+1,(min+col)]
    FilteredDataSet[i+1,(min+col)] <- ay
  }
  else
  {
```

```
        print("it didn't move")
        print(i)
        print(j)
      }
    }
    else if (i!=1)
    {
      if (FilteredDataSet[i-1,j]!=0)
      {
        if (abs(FilteredDataSet[i,j]-FilteredDataSet[i-1,j])<dxmax &
            abs(FilteredDataSet[i,j]-FilteredDataSet[i-1,j])> dxmin &
            abs(FilteredDataSet[i,(j+col)]-FilteredDataSet[i-1,(j+col)])<dymax)
        { # this is how the last point of the trajectory is checked and preserved
          print("last point or single point detected")
          print(i)
          print(j)
        }
        else
        {
          FilteredDataSet[i,j] <- 0
          FilteredDataSet[i,(j+col)] <- 0
        }
      }
      else
      {
        FilteredDataSet[i,j] <- 0
        FilteredDataSet[i,(j+col)] <- 0
      }
    }
    else
    {
      FilteredDataSet[i,j] <- 0
      FilteredDataSet[i,(j+col)] <- 0
    }
  }
  else
  {
    FilteredDataSet[i,j] <- 0
    FilteredDataSet[i,(j+col)] <- 0
  }
  }
  }
  info <- cbind(dxmax, dxmin, dymax)
  return(list(FilteredDataSet, info))
  }

#####

# For convinience in ploting we make a long form of the matrix

MakeLongForm <- function(InputMatrix) {
  LongForm <- data_frame(t = rep(1:(nrow(InputMatrix)), (ncol(InputMatrix)/2)),
    x = InputMatrix[, 1:(ncol(InputMatrix)/2)] %>% unlist() %>%
```

```
      as.vector(), y = InputMatrix[, (ncol(InputMatrix)/2 +
      1):ncol(InputMatrix)] %>% unlist() %>% as.vector(),
    z = rep(LETTERS[1:(ncol(InputMatrix)/2)], each = nrow(InputMatrix)))
  return(LongForm)

}

#####

PlotFX1 <- function(InputLongForm)
{
  ReadyPlot <- InputLongForm %>% ggplot(aes(x=t, y=x, colour=z)) + geom_line()
}

#####

PlotFX2 <- function(InputLongForm1, InputLongForm2) {
  ReadyPlot <- InputLongForm1 %>% ggplot(aes(x = t, y = x,
    colour = z)) + geom_line() + geom_point(data = dplyr::semi_join(InputLongForm2,
    InputLongForm1, by = c("x", "y")), alpha = 0.5)
  }

#####

PlotFX3 <- function(InputLongForm1, InputLongForm2, InputLongForm3) {
  ReadyPlot <- InputLongForm1 %>% ggplot(aes(x = t, y = x,
    colour = z)) + geom_line() + geom_point(data = dplyr::semi_join(InputLongForm2,
    InputLongForm1, by = c("x", "y")), alpha = 0.5) +
    geom_point(data = dplyr::anti_join(InputLongForm3,
    InputLongForm2, by = c("x", "y")), color = "black", alpha = 0.5)
}

#####

PlotXY1 <- function(InputLongForm1) {
  ReadyPlot <- InputLongForm1 %>% ggplot(aes(x = x, y = y,
    colour = z)) + geom_point() + theme_bw() + coord_cartesian(xlim = c(1000,
    8000), ylim = c(500, 3000)) + coord_equal(ratio = 0.1)

}

#####

PlotXY1FrameColor <- function(InputLongForm1) {
  ReadyPlot <- InputLongForm1 %>% ggplot(aes(x = x, y = y,
    colour = t)) + geom_point(size = 0.1) + theme_bw() +
    coord_cartesian(xlim = c(1000, 8000), ylim = c(500, 3000)) +
    xlab("X[nm]") + ylab("Y[nm]") + coord_equal(ratio = 1) +
    theme(axis.text = element_text(size = 12), axis.title = element_text(size = 14,
      face = "bold"))

}
```

```
#####

PlotXY2 <- function(InputLongForm1, InputLongForm2) {
  ReadyPlot <- InputLongForm1 %>% ggplot(aes(x = x, y = y)) +
    geom_point(size = 0.2, color = "red", alpha = 0.3) +
    geom_point(data = dplyr::anti_join(InputLongForm2, InputLongForm1,
      by = c("x", "y")), color = "gray", alpha = 0.2, size = 0.1) +
    theme_bw() + coord_cartesian(xlim = c(1000, 8000), ylim = c(500,
    3000)) + coord_equal(ratio = 1) + xlab("X[nm]") + ylab("Y[nm]") +
    theme(axis.text = element_text(size = 12), axis.title = element_text(size = 14,
      face = "bold"))
}

#####

PlotXY3 <- function(InputLongForm1, InputLongForm2, InputLongForm3) {
  ReadyPlot <- InputLongForm1 %>% ggplot(aes(x = x, y = y)) +
    geom_point(size = 1.2, color = "black", alpha = 1) +
    geom_point(data = dplyr::anti_join(InputLongForm2, InputLongForm1,
      by = c("x", "y")), color = "red", alpha = 0.1, size = 0.1) +
    geom_point(data = dplyr::anti_join(InputLongForm3, InputLongForm2,
      by = c("x", "y")), color = "gray", alpha = 0.1, size = 0.1) +
    theme_bw() + coord_cartesian(xlim = c(1000, 8000), ylim = c(500,
    3000)) + coord_equal(ratio = 1) + xlab("X[nm]") + ylab("Y[nm]") +
    theme(axis.text = element_text(size = 12), axis.title = element_text(size = 14,
      face = "bold"))
}

#####

# Here based on the input for min number of frames and min traveled distance,
# trajectories are arranged and information for macro in extracted

ExtractTrajectory <- function(DetectedLongForm, xRange, FrameLength)
{
  k=1
  n=1
  m=1
  f=0
  t=0
  x=0
  y=0
  z=0
  StartId <- 0
  EndId <- 0
  StartFrame <- 0
  EndFrame <- 0
  status <- ""
  TrajStart <- 0
  TrajEnd <- 0
  TrajStatus <- ""
  TrajInfo <- 0
  TrajNumber <- 0
```

```
for(i in 1:(nrow(DetectedLongForm)))
{# arranging the trajectories to be filtered
  if (!is.na(DetectedLongForm$x[i]))
  {
    f[k] <- k
    t[k] <- DetectedLongForm$t[i]
    x[k] <- DetectedLongForm$x[i]
    y[k] <- DetectedLongForm$y[i]
    z[k] <- DetectedLongForm$z[i]


    k <- k+1
  }
  else if (i!=nrow(DetectedLongForm) &
           is.na(DetectedLongForm$x[i]))
  {
    if (!is.na(DetectedLongForm$x[i+1]))
    {
      f[k] <- k
      t[k] <- DetectedLongForm$t[i]
      x[k] <- DetectedLongForm$x[i]
      y[k] <- DetectedLongForm$y[i]
      z[k] <- DetectedLongForm$z[i]

      StartFrame[m] <- DetectedLongForm$t[i]
      StartId[m] <- k
      m <- m+1
      k <- k+1

    }
    if (i!=1)
    {
      if (!is.na(DetectedLongForm$x[i-1]))
      {
        f[k] <- k
        t[k] <- DetectedLongForm$t[i]
        x[k] <- DetectedLongForm$x[i]
        y[k] <- DetectedLongForm$y[i]
        z[k] <- DetectedLongForm$z[i]

        EndFrame[n] <- DetectedLongForm$t[i]
        EndId[n] <- k
        n <- n+1
        if (StartFrame[1]==0)
        {
          StartFrame[1] <- 1
          StartId[1] <- 1
          m <- 2
        }
        k <- k+1
      }
    }
```

```
      else
      {
        print("do nothing")
      }
    }
  }
if (is.na(EndFrame[m-1]))
{
  EndFrame[m-1] <- max(DetectedLongForm$t)
  EndId[m-1] <- k-1
}

 EndFrame[EndFrame==1] <- max(DetectedLongForm$t)
 FrameCut <- data.frame(StartFrame, EndFrame)
 ExtractDet <- data.frame(f,t,x,y,z)


 k=1
 ToMacroStart <- ""
 ToMacroEnd <- ""
 Traj=0
 Extract=NA

 for (i in 1:nrow(FrameCut))
 {
   if(FrameCut$EndFrame[i]-FrameCut$StartFrame[i] > FrameLength)
   {# filter shortlived noise
     Traj <- subset(ExtractDet, f >= StartId[i] & f <= EndId[i])
     xx <- Traj[,3]
     if((max(xx[!is.na(xx)])-min(xx[!is.na(xx)])) > xRange)
     {# Filtering stationary trajectories
     r <- as.factor(rep("Yes", nrow(Traj)))
     status <- "Yes"
     }
     else
     {
     r <- as.factor(rep("No", nrow(Traj)))
     status <- "No"
     }
     s <- as.factor(rep(k, nrow(Traj)))
     Traj <- cbind(Traj,s,r)
     Extract <- rbind(Extract,Traj)
     TrajStart[k] <- StartFrame[i]
     TrajEnd[k] <- EndFrame[i]
     TrajNumber[k] <- k
     TrajStatus[k] <- status


     k <- k+1
   }
 }

 TrajInfo1 <- data.frame(TrajNumber,TrajStart,TrajEnd, TrajStatus)
```

```
  TrajInfo2 <- cbind(xRange,FrameLength)
  return(list(Extract,TrajInfo1,TrajInfo2))

}



#####

# Function for spatail filtering

FilterIt <- function(RawData, xmin, xmax, ymin, ymax, intenmax,
  intenmin) {
  filter <- SourceData %>% filter(`x [nm]` > xmin, `x [nm]` <
    xmax, `y [nm]` > ymin, `y [nm]` < ymax, `intensity [photon]` >
    intenmin, `intensity [photon]` < intenmax)
  info <- cbind(xmin, xmax, ymin, ymax, intenmax, intenmin)
  return(list(filter, info))
}

#####

PlotFX1TrajOnSeq <- function(InputLongForm) {
  ReadyPlot <- InputLongForm %>% ggplot(aes(x = seq(1, nrow(InputLongForm),
    1), y = x, colour = s)) + geom_line()
}

#####

PlotFX1TrajOnFra <- function(InputLongForm) {
  ReadyPlot <- InputLongForm %>% ggplot(aes(x = t, y = x, colour = s)) +
    xlab("Frame Number") + geom_line() + theme(axis.text.x = element_text(size = 12),
    axis.title.y = element_text(size = 12))
}

#####

PlotFX1SingleTraj <- function(InputLongForm1, InputLongForm2,
  TrajNumber) {
  fil <- InputLongForm1 %>% filter(s == TrajNumber)
  at <- max(fil$t)
  bt <- min(fil$t)
  ax <- max(fil$x[!is.na(fil$x)])
  bx <- min(fil$x[!is.na(fil$x)])


  ReadyPlot <- fil %>% ggplot(aes(x = t, y = x)) + geom_line() +
    geom_point(data = dplyr::semi_join(InputLongForm2, InputLongForm1,
    by = c("x", "y")), alpha = 0.5) +
    geom_point(data = dplyr::anti_join(InputLongForm2,
    InputLongForm1, by = c("x", "y")), alpha = 0.3, color = "green") +
    coord_cartesian(xlim = c(bt, at), ylim = c(bx, ax))
}
```

```
#####

PlotFX1SingleTraj2 <- function(InputLongForm,  TrajNumber, Name)
{
  fil <- InputLongForm %>%filter(s==TrajNumber & n==Name)
  at <- max(fil$t)
  bt <- min(fil$t)
  ax <- max(fil$x[!is.na(fil$x)])
  bx <- min(fil$x[!is.na(fil$x)])


  ReadyPlot <- fil%>% ggplot(aes(x=t, y=x)) + geom_line()+
    geom_point(aes(x=t, y=x), alpha=0.5)+
    coord_cartesian(xlim=c(bt,at), ylim = c(bx,ax))
}


#####

PlotFX1YesTraj <- function(InputLongForm)
{
  fil <- InputLongForm %>%filter(v=="Yes")
  ReadyPlot <- fil%>%
  ggplot(aes(x=seq(1,nrow(fil),1), y=x, colour=s)) +
  geom_line()
}

#####

PlotFX1YesTrajOnFrame <- function(InputLongForm)
{
  fil <- InputLongForm %>%filter(v=="Yes")
  ReadyPlot <- fil%>% ggplot(aes(x=t, y=x, colour=s)) +
  geom_line()
}

#####

readinteger <- function(What)
{
  n <- readline(prompt=What)
  return(as.integer(n))
}
```

3.3 Extracting frames containing trajectories

The output of the code in the last step is the address to the detected trajectories (in terms of frame number) that later will be used to extract the frames in which the interaction is observed.

Here is the script that was used to extract the frames of the events and concatenate them one after another in FIJI. The outputs of this script are the tiff files that contain all events (trajectories) registered around the DNA.

```
FileName = "EndoV_150mMNaCl_2016101704_7.5mspf_112nmpp_1th_1000X"
open("tiff\\"+FileName+".tif");
pathfile="FrameCuts\\"+FileName+".txt";
filestring=File.openAsString(pathfile);
rows=split(filestring, "\n");
StartF=newArray(rows.length);
EndF=newArray(rows.length);
for(i=1; i<rows.length; i++){
columns=split(rows[i],"\t");
StartF[i]=parseInt(columns[2]);
EndF[i]=parseInt(columns[3]);
}
name=0
list="image1= Concatenated Stacks"
open("Separator.tif");
for (i=1; i<rows.length; i++) {
if(i==1)
{
selectWindow(FileName+".tif");
slices=""+(StartF[i])+"-"+(EndF[i]);
  run("Make Substack...", "  slices=" + slices);
  run("Label...", "format=0 starting=1 interval=1 x=0
      y=10 font=9 text=T"+i);
name= "image1=[Substack ("+slices+")] image2=Separator.tif";
run("Concatenate...", "  title=[Concatenated Stacks]"+name);
print(i);

      }  else {
open("Separator.tif");
selectWindow(FileName+".tif");
slices=""+(StartF[i])+"-"+(EndF[i]);
  run("Make Substack...", "  slices=" + slices);
  run("Label...", "format=0 starting=1 interval=1 x=0
      y=10 font=9 text=T"+i);
name= "image1=[Concatenated Stacks] image2=[Substack ("+slices+")]
      image3=Separator.tif";

run("Concatenate...", "  title=[Concatenated Stacks]"+name);
print(i);
}
}
saveAs("tiff", "FrameCuts\\"+FileName);
```

4 Data transformation

4.1 Rearranging the raw data

The result of the image processing and tracking code were two RDS files containing the bead positions and the trajectories information for each protein (only 6 data sets among 92). Using following script these information are combined into a single data set that is ready for further analysis.

From this step onward the outputs of all steps are included in a folder titled “Processed”. All information for

running the code and producing the PDF report is provided in this folder.

```
## Import all available data
SubStep = "1.1."
time1.1 = ".2017-04-07"

if (!file.exists(paste("Processed/", SubStep, "SignalWithBeadPosition",
  time1.1, ".rds", sep = ""))) {
  # print(paste('Step', SubStep,': Analysis started!')) process
  # from scratch
  path <- "Processed/"

  files <- list.files(path, pattern = ".rds") %>% grep("signal",
    ., value = TRUE, ignore.case = TRUE, invert = FALSE)

  for (file in files) {
    if (file == files[1])
      dat <<- readRDS(file.path(path, file)) %>% tbl_df()
    if (file != files[1])
      dat <<- bind_rows(dat, readRDS(file.path(path, file)))
  }

  dat %<>% as_data_frame() %>% mutate(uniqueID = n) %>% separate(n,
    c("enzyme", "NaCl", "date", "frame_interval", "pixel_size",
      "id", "dilution"), sep = "_")

  # fixing the concentrations on 1or100 and 50or150
  dat %<>% mutate(NaCl = stringr::str_replace(NaCl, "mMNaCl",
    ""), NaCl = stringr::str_replace(NaCl, "or100", ""),
    NaCl = stringr::str_replace(NaCl, "or150", ""), NaCl = as.numeric(NaCl)) %>%
    mutate(frame_interval = stringr::str_replace(frame_interval,
      "mspf", "") %>%
        as.numeric()) %>%
    mutate(uniqueID2 = uniqueID) %>%
    unite(ID, s, uniqueID2) %>%
    mutate(ID = as.factor(ID) %>%
    as.numeric()) %>%
    dplyr::select(-z)


  # bead position data
  beadFiles <- list.files(path, pattern = ".rds") %>% grep("bead",
    ., value = TRUE, ignore.case = TRUE, invert = FALSE)

  for (file in beadFiles) {
    if (file == beadFiles[1])
      beads <<- readRDS(file.path(path, file)) %>%
        tbl_df()
    if (file != beadFiles[1])
      beads <<- bind_rows(beads, readRDS(file.path(path,
        file)))
  }

  beads %<>%
```

```
    as_data_frame() %>%
    mutate(uniqueID = stringr::str_replace(FileName,
    ".tif.xml", ""), t = FrameNumber)

  datWithBeadPositions <- left_join(dat, beads)

  # final tidy dataset
  datWithBeadPositions %<>%
    filter(!is.na(x)) %>%
    select(FrameNumber,
    x, y, i, r, v, ID, XPosition, YPosition, enzyme:uniqueID) %>%
    set_colnames(c("Frame_number", "X", "Y", "Intensity",
      "Delta_X > 300", "Visual_confirmation", "Unique_trajectory_ID",
      "Bead_X", "Bead_Y", "Enzyme", "NaCl", "Date", "Frame_interval",
      "Pixel_size", "Protein_batch_ID", "Dilution", "File_name"))

  saveRDS(datWithBeadPositions, paste("Processed/", SubStep,
    "SignalWithBeadPosition", time1.1, ".rds", sep = ""))

} else {
  # print(paste('Step', SubStep,': Data is loaded from the last
  # analysis on', time1.1, '!', sep = '')) datWithBeadPositions
  # <-
  # readRDS('Processed/Step1-SignalWithBeadPosition-2017-03-21.rds')
}
```

4.2 Filtering of the data sets

Some data sets do not have any trajectory recorded and in some cases there are two DNAs attached to one bead. Using this script the data sets without trajectories are filtered and those with two DNAs are separated into two data set for each DNA.

```
# Manual annotation
SubStep = "1.2."
time1.2 = ".2017-02-10"

if (!file.exists(paste("Processed/", SubStep, "DataSetClassification",
  time1.2, ".rds", sep = ""))) {
  # print('Step1.2.: Analysis started! You need to look for the
  # data sets with two DNA in the field of view!')
  datWithBeadPositions <- readRDS("Processed/Step1-SignalWithBeadPosition-2017-03-21.rds")
  separation_point = NULL
  classification = NULL
  for (name in (datWithBeadPositions$File_name %>% unique)) {
    plot <- datWithBeadPositions %>%
      filter(Visual_confirmation == "Yes", !is.na(Bead_X)) %>%
      filter(File_name == name) %>%
      mutate(centred_X = X - Bead_X, centred_Y = Y - Bead_Y,
        Theta = acos(centred_Y/centred_X)) %>% ggplot(aes(x = centred_X,
      y = centred_Y, colour = Theta)) +
      geom_point(alpha = 0.15) +
      viridis::scale_color_viridis(guide = FALSE) +
      geom_smooth(method = "lm") +
```

```
      coord_equal(ratio = 1)

    print(plot)

    x <- readline("Ignore or not? Hit return for not and I for Ignore")
    names(x) <- name
    x <- ifelse(x == "", "Simple", "Ignore")
    classification <- c(classification, x)

    y <- readline("If there are two DNAs enter their separation point on Y")

    y <- ifelse(y == "", NA, as.numeric(y))
    a <- c(name, y)
    separation_point <- rbind(separation_point, a)
  }

  classification %<>%
    data_frame(Analysis_required = ., File_name = names(.))

  # Analysis overview

  classification %<>%
    filter(!stringr::str_detect(File_name,"YOYO")) %>%
    left_join(., (datWithBeadPositions %>%
                    filter(!stringr::str_detect(File_name,"YOYO")) %>%
      group_by(File_name) %>%
        summarize(count = n(),
    framesInspected = sum(Visual_confirmation == "Yes"),
    framesUninspected = sum(Visual_confirmation == "No"),
    match = (framesUninspected + framesInspected) == count))) %>%
    mutate(Analysis_required = ifelse(framesInspected ==
      0, "Ignore", Analysis_required))

  separation_point <- data.frame(separation_point)
  names(separation_point) <- (c("File_name", "separation_point"))

  classification <- left_join(classification, separation_point)

  saveRDS(classification, paste("Processed/", SubStep, "DataSetClassification",
    time1.2, ".rds", sep = ""))
} else {
  # classification <-
  # readRDS('Processed/Step2-DataSetClassification-2017-02-10.rds')
  # print(paste('Step', SubStep,': Data is loaded from the last
  # analysis on', time1.2, '!', sep = ''))
}
```

4.3 Trajectory alignment

In this step we:

1. adjust the end of DNA at bead position.

2. apply the rotation matrix to rotate the trajectories so that the displacements are mostly projected into the X-direction.

3. make a threshold for the fluctuations in Y-direction.

In Fig. 6,7 and 8 the traces of the detected trajectories for different data sets are coloured by whether they are within the Y-fluctuation threshold or not.

```
SubStep = "1.3."
time1.3 = ".2017-04-07"

if (!file.exists(paste("Processed/", SubStep, "AlignedData",
  time1.3, ".rds", sep = ""))) {
  # print(paste('Step', SubStep,': Analysis started!'))
  datWithBeadPositions <- readRDS(paste("Processed/1.1.SignalWithBeadPosition",
    time1.1, ".rds", sep = ""))
  classification <- readRDS(paste("Processed/1.2.DataSetClassification",
    time1.2, ".rds", sep = ""))

  # introduction of the rotation matrix
  rotatePoints <- function(xPos, yPos, estimate) {
    coordinates <- matrix(c(xPos, yPos), ncol = 1)
    mat <- matrix(c(cos(atan(estimate)), sin(atan(estimate)),
      -sin(atan(estimate)), cos(atan(estimate))), ncol = 2)
    (mat %*% coordinates) %>% as.vector()
  }

  # joining the classification with the mian data set
  datWithBeadPositions <- left_join(datWithBeadPositions, (classification %>%
    select(-count, -framesUninspected, -match)))
  # remove YOYO and empty bead position data
  datWithBeadPositions %<>% filter(!is.na(Bead_X))

  # centre on beads
  datWithBeadPositions %<>% mutate(centred_X = X - Bead_X,
    centred_Y = Y - Bead_Y)

  # regroup
  splits <- datWithBeadPositions %>%
    mutate(separation_point = ifelse(is.na(separation_point),
    -10000, separation_point)) %>%
    group_by(Unique_trajectory_ID) %>%
    mutate(aboveSeparation = sign(centred_Y - separation_point)) %>%
    summarize(grouping = median(aboveSeparation) > 0) %>%
    mutate(appendToFileName = ifelse(grouping, "", "_A"))

  datWithBeadPositions <- left_join(datWithBeadPositions, splits) %>%
    mutate(File_name = paste(File_name, appendToFileName, sep = ""))


  rotationMatricesUnseparated <- datWithBeadPositions %>%
    filter(Visual_confirmation == "Yes", is.na(separation_point)) %>%
    group_by(File_name) %>%
    do(tidy(lm(centred_Y ~ centred_X + 0, data = .)))
```

```
 rotationMatricesSeparated <- datWithBeadPositions %>%
   filter(Visual_confirmation == "Yes", !is.na(separation_point)) %>%
   group_by(File_name) %>%
   do(tidy(lm(centred_Y ~ centred_X, data = .)))

 rotationMatrices <- bind_rows(rotationMatricesUnseparated,
   rotationMatricesSeparated) %>%
   filter(term == "centred_X")

 datWithBeadPositionsRotated <- datWithBeadPositions %>%
   left_join(.,rotationMatrices) %>%
   rowwise %>%
   mutate(corrected_X = rotatePoints(centred_X,
   centred_Y, -estimate)[1], corrected_Y = rotatePoints(centred_X,
   centred_Y, -estimate)[2]) %>%
   ungroup()

 ## Maximum Y fluctuation is obtained from the visually inspected data

datWithBeadPositionsRotated %>%
   filter(Visual_confirmation == "Yes") %>%
 group_by(Unique_trajectory_ID) %>%
   summarise(sd = sd(corrected_Y)) %>%
 summarise(max(sd))

 onDNAthresholds <- datWithBeadPositionsRotated %>%
   dplyr::filter(Analysis_required == "Simple",
   Visual_confirmation == "Yes") %>%
   group_by(File_name) %>%
   summarize(mid_y = mean(corrected_Y, trim = 0.5)) %>%
   mutate(upperCutoff = mid_y + 200, lowerCutoff = mid_y - 200)

 # ON DNA threshold set at 200 here

 datWithBeadPositionsRotated %<>%
   left_join(., onDNAthresholds) %>%
   mutate(nudged_Y = corrected_Y - mid_y, On_DNA = abs(nudged_Y) < 200)

 datWithBeadPositionsRotated %<>%
   filter(Analysis_required == "Simple")

 # check the y fluctuation of protein trajectories on DNA

 datWithBeadPositionsRotated%>%
   filter(`Delta_X > 300`== "Yes", On_DNA) %>%
   group_by(Enzyme, Unique_trajectory_ID) %>%
 summarise(sdX= sd(corrected_X), sdY= sd(corrected_Y)) %>%
 ungroup() %>%
 filter(!is.na(sdX)) %>%
 group_by(Enzyme) %>%
 summarise(Y.fluctuation = mean(sdY))

 # compare with y fluctuation of proteins stuck to the surface
```

```
  datWithBeadPositionsRotated%>%
    filter(`Delta_X > 300`== "No", !On_DNA) %>%
    group_by(Enzyme, Unique_trajectory_ID) %>%
  summarise(sdX= sd(corrected_X), sdY= sd(corrected_Y), length = n()) %>%
  ungroup() %>% filter(length > 100) %>%
  filter(!is.na(sdX)) %>%
  group_by(Enzyme) %>%
  summarise(Y.fluctuation = mean(sdY))

  # trajectories on DNA fluctuate in y around twice as much as proteins stuck to
  # the sruface

  saveRDS(datWithBeadPositionsRotated, paste("Processed/",
    SubStep, "AlignedData", time1.3, ".rds", sep = ""))
  } else {
  datWithBeadPositionsRotated <- readRDS(paste("Processed/",
    SubStep, "AlignedData", time1.3, ".rds", sep = ""))

  }



# These are the outputs of this step

datWithBeadPositionsRotated %>%
  filter(Enzyme == "EndoV", `Delta_X > 300` =="Yes") %>%
  ggplot(aes(x = corrected_X, y = nudged_Y, colour = On_DNA)) +
  geom_point(alpha = 0.9, size = 0.1) +
  viridis::scale_color_viridis(discrete = TRUE, guide = FALSE) +
  coord_equal(ratio = 1) +
  facet_wrap(~File_name) +
  ggtitle("EndoV") + ylab("Y (nm)") + xlab("X (nm)")

ggsave("EndoVFiltered.png", path =  "data_transformation/",
       dpi = 200)

datWithBeadPositionsRotated %>%
  filter(Enzyme == "mEndoV", `Delta_X > 300` == "Yes") %>%
  ggplot(aes(x = corrected_X, y = nudged_Y, colour = On_DNA)) +
  geom_point(alpha = 0.9, size = 0.1) +
  viridis::scale_color_viridis(discrete = TRUE, guide = FALSE) +
  coord_equal(ratio = 1) +
  facet_wrap(~File_name) +
  ggtitle("mEndoV") + ylab("Y (nm)") + xlab("X (nm)")

ggsave("mEndoVFiltered.png", path= "data_transformation/",
       dpi = 200)

datWithBeadPositionsRotated %>%
  filter(Enzyme == "hOgg1", `Delta_X > 300` ==  "Yes") %>%
  ggplot(aes(x = corrected_X, y = nudged_Y, colour = On_DNA)) +
  geom_point(alpha = 0.9, size = 0.1) +
  viridis::scale_color_viridis(discrete = TRUE, guide = FALSE) +
```

```
  coord_equal(ratio = 1) +
  facet_wrap(~File_name) +
  ggtitle("hOgg1") + ylab("Y (nm)") + xlab("X (nm)")

ggsave("hOGG1Filtered.png", path= "data_transformation/",
       dpi = 200)
```

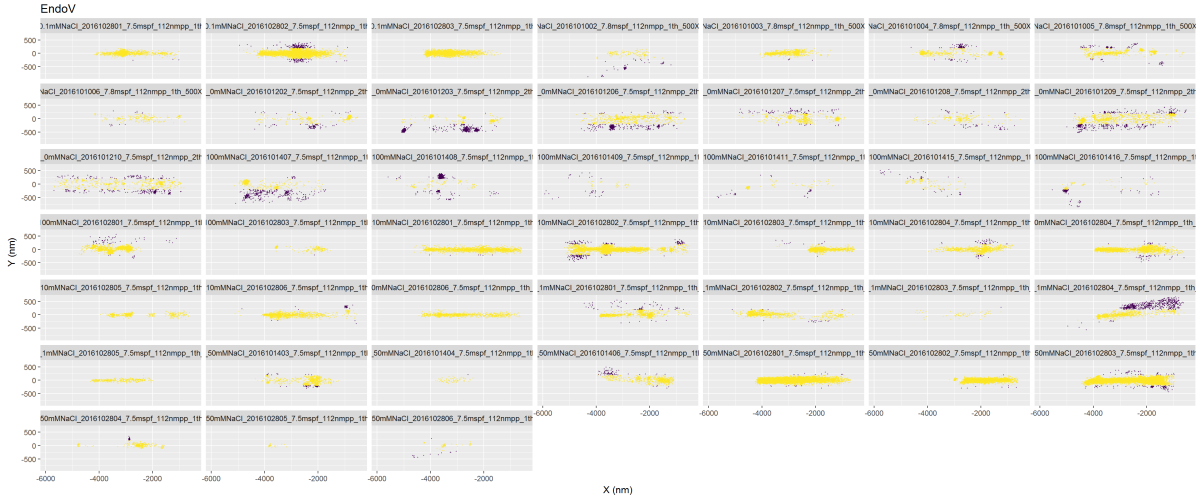

Figure 6: EndoV

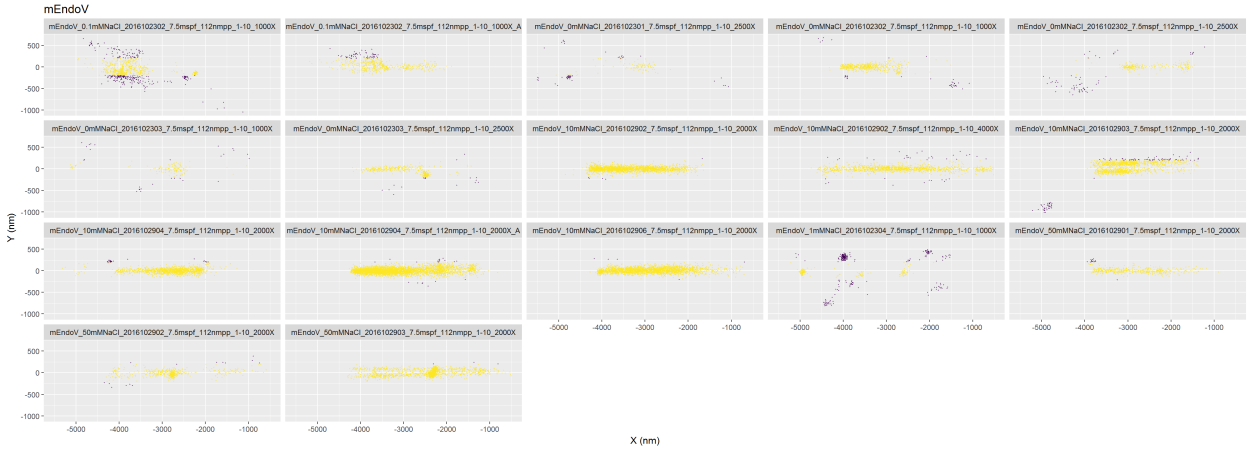

Figure 7: wedge mutant EndoV

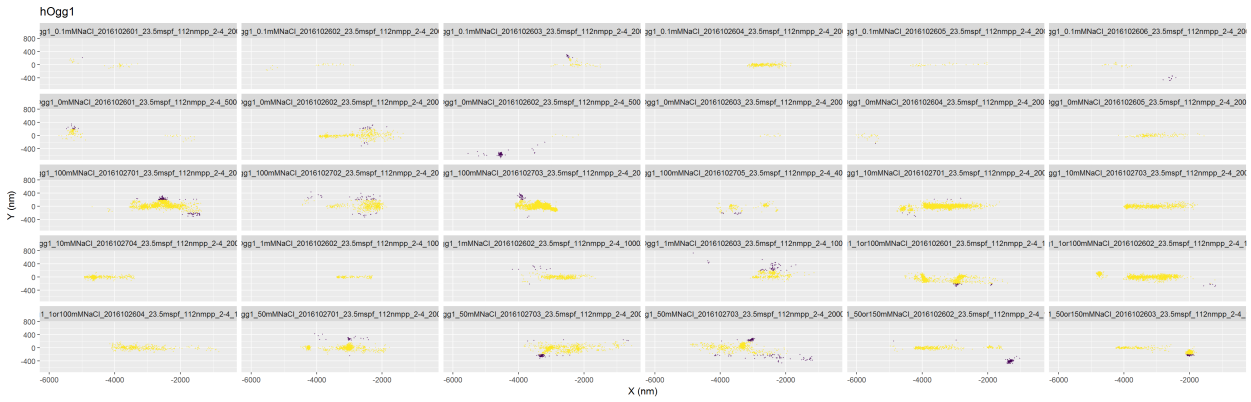

Figure 8: hOGG1

5 Blinking correction and binding lifetime

5.1 Blinking correction

The dyes used in these experiments can go to off state for a very short period of time (blinking). The tracking code separates the trajectories after each blinking event, in this part we correct this effect using following script.

```
SubStep = "2.1."
time2.1 = ".2017-04-07"

if (!file.exists(paste("Processed/", SubStep, "AlignedDataBlinkingCorrected",
  time2.1, ".rds", sep = ""))) {
  # print(paste('Step', SubStep,': Analysis started!'))
  datWithBeadPositionsRotated <- readRDS(paste("Processed/1.3.AlignedData",
    time1.3, ".rds", sep = ""))

  # Test that the all frames are contiguous within a
  # trajectory, this part does not to be ran everytime, only
  # when data is revised

  CorrectedBlinking <- datWithBeadPositionsRotated %>%
    mutate(Trj = lead(Unique_trajectory_ID,
    n = 1) - Unique_trajectory_ID, frm = lead(Frame_number,
    n = 1) - Frame_number, Crx = lead(corrected_X, n = 1) -
    corrected_X)

  CorrectedBlinking %<>%
    mutate(Trj = ifelse(Trj != 0 & frm == 2 & !is.na(Crx) & Crx < 600, 0, Trj),
           trjNew = sign(Trj)) %>%
    mutate(trjNew = abs(lag(trjNew))) %>%
    ungroup %>%
    mutate(trjNew = ifelse(is.na(trjNew), 0, trjNew), grp = cumsum(trjNew)) %>%
    group_by(grp) %>%
```

```
    mutate(New_Unique_trajectory_ID = Unique_trajectory_ID[1],
      New_Delta_X = ifelse(sum(`Delta_X > 300` == "No") == n(), "No", "Yes"),
      New_Visual_confirmation = ifelse(sum(Visual_confirmation ==
        "No") == n(), "No", "Yes"))

  CorrectedBlinking %>%
    group_by(Enzyme, NaCl) %>%
    summarize(trajectory_count = length(unique(Unique_trajectory_ID)),
              totalFrames = n())

  saveRDS(CorrectedBlinking, paste("Processed/", SubStep, "AlignedDataBlinkingCorrected",
    time2.1, ".rds", sep = ""))

} else {
  # print(paste('Step', SubStep,': Data is loaded from the last
  # analysis on', time2.1, '!', sep = ''))
  CorrectedBlinking <- readRDS(paste("Processed/", SubStep,
    "AlignedDataBlinkingCorrected", time2.1, ".rds", sep = ""))
}
```

5.2 Binding lifetime

The binding life time is defined as the duration of uninterrupted binding and scanning events. The binding lifetime of different proteins in different salt concentrations is shown in Fig.9.

```
trajectoryLengths <- CorrectedBlinking %>%
  group_by(Enzyme, NaCl, New_Unique_trajectory_ID) %>%
  filter(New_Visual_confirmation == "Yes") %>%
  summarize(numFrames = n(),
            duration = numFrames * unique(Frame_interval)/1000) %>%
  arrange(desc(numFrames))


trajectoryLengths$Enzyme <-  factor(trajectoryLengths$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))

trajectoryLengths %>%
  filter(NaCl>0) %>%
  ggplot(aes(x=NaCl, y=duration, colour=Enzyme)) +
  geom_point(size=1) +
  geom_line(data=trajectoryLengths %>%
              summarize(mean_duration=mean(duration), weight=n()) %>%
              filter(NaCl>0), aes(y=mean_duration, size=weight))+
  geom_point(data=trajectoryLengths %>%
               summarize(mean_duration=mean(duration)) %>%
               filter(NaCl>0), aes(y=mean_duration), size=5) +
   scale_color_discrete(name = "Enzyme",
                        labels= c("hOGG1", "wt-EndoV", "wm-EndoV"))+
  coord_cartesian(ylim = c(0, 1.5)) +
  scale_x_log10()+
  xlab("NaCl (mM)")+
  ylab("Binding life time (s)")
```

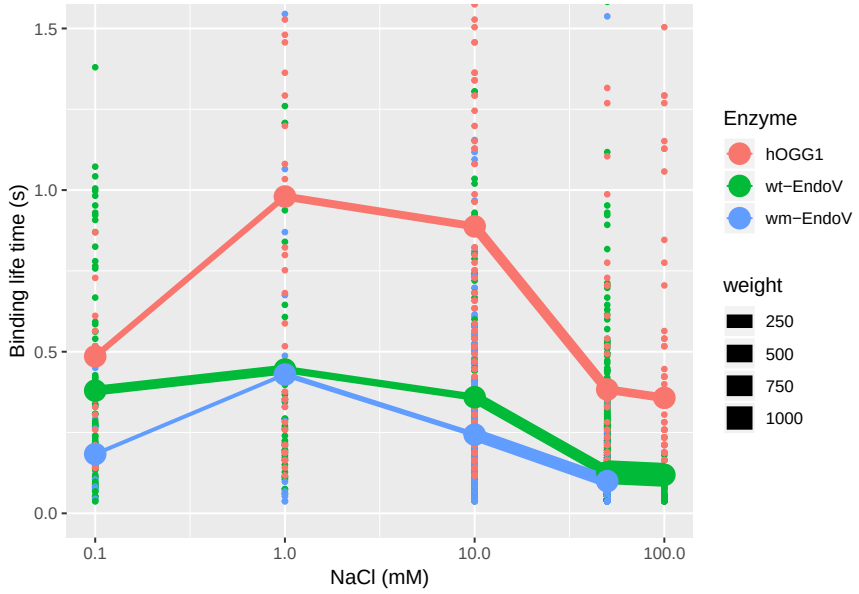

Figure 9: Binding lifetime

6 Instantaneous diffusion rate

6.1 Local MSD function

In order to calculate the local MSD of the trajectories we built a function which runs through the data and calculate the instantaneous diffusion rate with moving window of 5, 7 and 10 steps. This function is written and controlled in R but it will be compiled with C++ using Rcpp package for higher time efficiency.

```
#include <Rcpp.h>
using namespace Rcpp;

// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar). Learn
// more about Rcpp at:
//
//   http://www.rcpp.org/
//   http://adv-r.had.co.nz/Rcpp.html
//   http://gallery.rcpp.org/
//

// [[Rcpp::export]]
NumericVector localMSDcomplete(NumericVector positions, int windowSize) {

  int n = positions.size();

  if (windowSize >= n) {
    windowSize = n - 1;
  }

  // container for results
```

```
NumericMatrix mat(n-1, windowSize-1);

// container for mean square displacements
NumericVector msds(n);
// calculate square displacement at each distance
for(int j = 1; j < windowSize; j++) {

  // container for square displacements internally
  // NumericVector msds_internal(n-j);

  for(int i = 0; i < n-j; i++) {

    double value = 0;
    int kk = 0;

    // max aggregation windowlength
    int aggWindowLength = windowSize-1;
    if (n-i-j < windowSize-1){
      aggWindowLength = n-i-j;
    }


    // aggregate square displacements
    for(int k = 0; k<aggWindowLength; k++) { //windowSize-1
      // local difference
      double tmp = positions(i+k) - positions(i+j+k);
      // aggregated square values
      value = value + tmp * tmp;

      kk = k;
    }

    // mean of values
    value = value/(kk+1);


    mat(i,j-1) = value / j;

    // msds_internal(i) = value;
  }
  // msds(j-1) = mean(msds_internal) ;
}

for (int j = 0; j <= n - windowSize; j++) {
  msds(j) = mean(mat(j,_));
}

// NumericVector output(1);

// output(0) = mean(msds);


return(msds);
```

```
}


// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.
//

/*** R
# library(tidyverse)
# library(magrittr)

localMSDcomplete(1:20, 6)
  set.seed(10000)
  localMSDcomplete(cumsum(rnorm(30)), 6)
  set.seed(10000)
  rnorm(30) %>% cumsum() %>% diff() %>% magrittr::raise_to_power(., 2)
  */
```

6.2 Instantaneous diffusion rate distribution

Using the Rccp function the Instantaneous diffusion rate is calculated and shown for different width of the moving window in Fig.10-12. In these figure it is shown that modality of scanning is independent of the width of the moving window.

```
if (!file.exists("Processed/localMSD_real_data.rds"))
{

Rcpp::sourceCpp('sourcesCpp/msdsComplete.cpp')

aligned.data <-
  readRDS("Processed/2.1.AlignedDataBlinkingCorrected.2017-04-07.rds")


tidy.data <- aligned.data %>%
  ungroup()%>% select(File_name, Unique_trajectory_ID, New_Unique_trajectory_ID,
                                 Frame_number, corrected_X, corrected_Y,
                                 Enzyme, NaCl, Frame_interval, On_DNA,
                                  Visual_confirmation, `Delta_X > 300`) %>%
    arrange(Unique_trajectory_ID, Frame_number)

## Number of trajectories

tidy.data %>% group_by(Enzyme) %>% filter(`Delta_X > 300`== "Yes", On_DNA) %>%
  summarize(trajectory_count=length(unique(New_Unique_trajectory_ID)),
            totalFrames=n())

## localMSDs are given in nm2/frame we need D in terms of  micrometer2/s
## therefore we divid the values of MSDs by 2000 * frame.interval

localMSDs <- tidy.data %>%
  group_by(Unique_trajectory_ID) %>%
  mutate(time=Frame_number-min(Frame_number)) %>%
```

```
  mutate(localMSD_05 = localMSDcomplete(corrected_X, 5)/ (2000*Frame_interval),
         localMSD_07 = localMSDcomplete(corrected_X, 7)/ (2000*Frame_interval),
         localMSD_10 = localMSDcomplete(corrected_X, 10)/ (2000*Frame_interval),
         localMSD_15 = localMSDcomplete(corrected_X, 15)/
           (2000*Frame_interval)) %>%
ungroup()

localMSDs$Enzyme <-  factor(localMSDs$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))
saveRDS(localMSDs, "Processed/localMSD_real_data.rds")
}else{

localMSDs <- readRDS("Processed/localMSD_real_data.rds")
}

labeloo3 <- c('EndoV'= "wt-EndoV" , 'hOgg1'= "hOGG1",
              'mEndoV'= "wm-EndoV")



localMSDs %>%
  filter(On_DNA, `Delta_X > 300`=="Yes") %>%
  filter(localMSD_05!=0) %>% group_by(NaCl) %>%
  ggplot(aes(x=localMSD_05)) +
  facet_wrap(~Enzyme, ncol = 3, scales = "free_y",
             labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100) +
  geom_vline(xintercept = 1.35)+
  scale_x_log10()+
  ggtitle("Moving Window:05") +
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2/s))) +
 geom_segment(data=data.frame(x=c(0.89,1.30,1.30),
                              y= c(0.075,0.075,0.075),
                              Enzyme=c("hOgg1","EndoV",
                                       "mEndoV")),
              aes(x= x, y= 0, xend= x ,yend=y), inherit.aes=FALSE, size= 0.6)
```

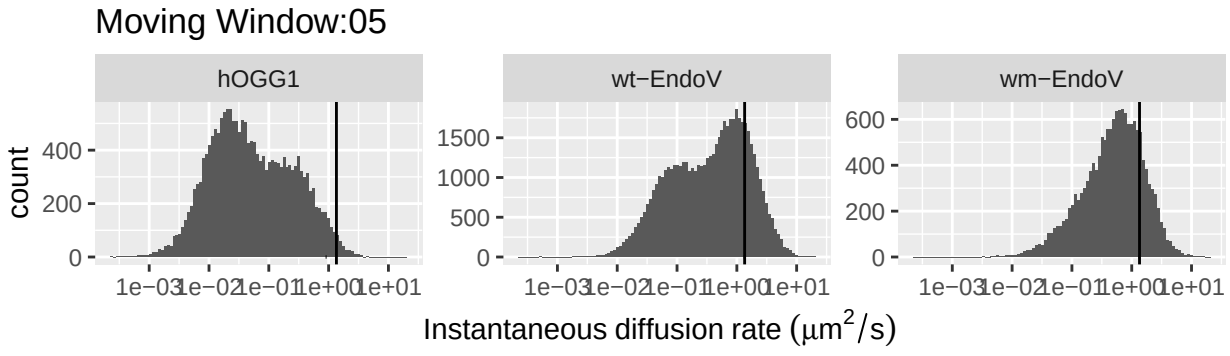

Figure 10: Instantaneous diffusion rate distribution

```
localMSDs %>%
  filter(On_DNA, `Delta_X > 300`=="Yes") %>%
  filter(localMSD_07!=0) %>%
  ggplot(aes(x=localMSD_07)) +
  facet_wrap(~Enzyme, ncol = 3, scales = "free_y",
             labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100) +
  geom_vline(xintercept = 1.35)+
  scale_x_log10()+
  ggtitle("Moving Window:07")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2/s))) +
  geom_segment(data=data.frame(x=c(0.89,1.30,1.30),
                               y= c(0.075,0.075,0.075),
                               Enzyme=c("hOgg1","EndoV",
                                        "mEndoV")),
               aes(x= x, y= 0, xend= x ,yend=y), inherit.aes=FALSE, size= 0.6)
```

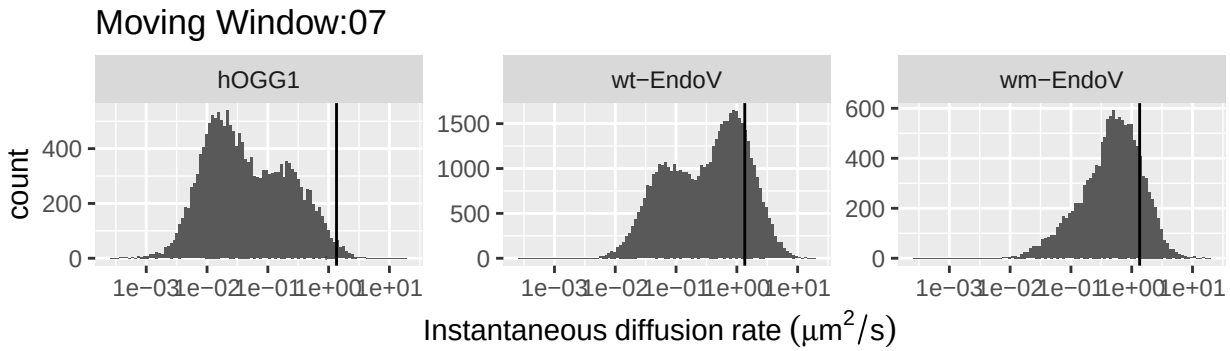

Figure 11: Instantaneous diffusion rate distribution

```
localMSDs %>%
  filter(On_DNA, `Delta_X > 300`=="Yes") %>%
  filter(localMSD_10!=0) %>%
  ggplot(aes(x=localMSD_10)) +
  facet_wrap(~Enzyme, ncol = 3, scales = "free_y",
             labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100) +
  geom_vline(xintercept = 1)+
  scale_x_log10()+
  ggtitle("Moving Window:10")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2/s))) +
  geom_segment(data=data.frame(x=c(0.89,1.30,1.30),
                               y= c(0.075,0.075,0.075),
                               Enzyme=c("hOgg1","EndoV",
                                        "mEndoV")),
               aes(x= x, y= 0, xend= x ,yend=y), inherit.aes=FALSE, size= 0.6)
```

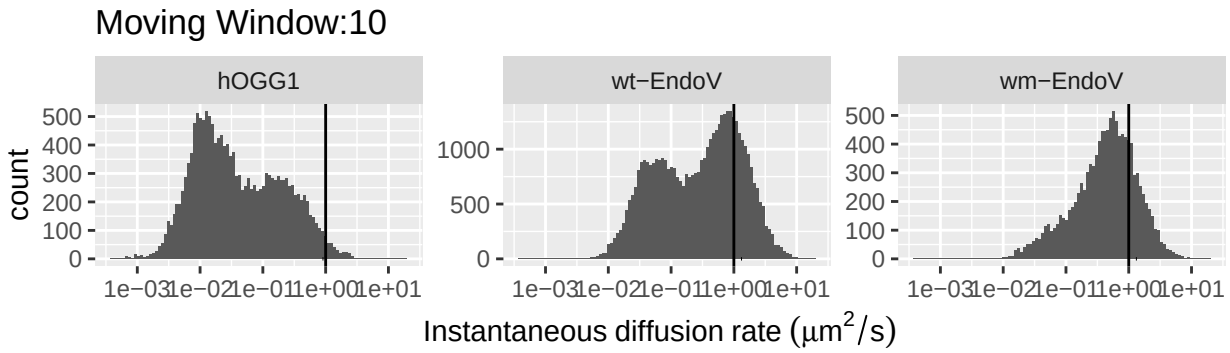

Figure 12: Instantaneous diffusion rate distribution

Now we calculate the percentage of trajectories that pass the upper theoretical limit of the diffusion rate for helical sliding(calculated in 9.3).

```
## % of trajectories passing the speed limit

## hOGG1
localMSDs %>%
  filter(Enzyme == "hOgg1", On_DNA, `Delta_X > 300`=="Yes") %>%
  mutate(passed = ifelse(localMSD_07 <0.89, 0,1)) %>%
  ungroup()%>%
  summarise(sum(passed, na.rm =T)/n())
```

```
## # A tibble: 1 x 1
##   `sum(passed, na.rm = T)/n()`
##                          <dbl>
## 1                       0.0244
```

```
## EndoV
localMSDs %>%
  filter(Enzyme == "EndoV", On_DNA, `Delta_X > 300`=="Yes") %>%
  mutate(passed = ifelse(localMSD_07 <1.3, 0,1)) %>%
  ungroup()%>%
  summarise(sum(passed, na.rm =T)/n())
```

```
## # A tibble: 1 x 1
##   `sum(passed, na.rm = T)/n()`
##                          <dbl>
## 1                        0.138
```

```
## mEnodV
localMSDs %>%
  filter(Enzyme == "mEndoV", On_DNA, `Delta_X > 300`=="Yes") %>%
  mutate(passed = ifelse(localMSD_07 <1.3, 0,1)) %>%
  ungroup()%>%
  summarise(sum(passed, na.rm =T)/n())
```

```
## # A tibble: 1 x 1
##   `sum(passed, na.rm = T)/n()`
##                          <dbl>
## 1                        0.121
```

6.3 Simulation of single-mode random walks

Here we want to simulate single-mode random walks with average diffusion rates equal to those of the proteins. The average diffusion rate for hOgg1, EndoV and mutant EndoV is calculated as $0.13\mu m^2/s$, $0.65\mu m^2/s$ and $0.70\mu m^2/s$ with time interval of $23.5ms$, $7.5ms$ and $7.5ms$ respectively.

In the next step the instantaneous diffusion rate distribution of the proteins are compared with those of their corresponding single-mode simulated random walks. The result is shown in Fig.13. The histogram overlaid with red density plots are the instantaneous diffusion rate distribution of the proteins and the solid black lines are the corresponding simulated single mode random walks.

```
localMSDs %>% filter(localMSD_15!=0, `Delta_X > 300` == "Yes", On_DNA)%>%
  group_by(Enzyme) %>%
  summarise(ave.diffusion.rate = mean(localMSD_15, na.rm = T),
            sd.diffusion.rate = sd(localMSD_15, na.rm = T))
```

```
## # A tibble: 3 x 3
##   Enzyme ave.diffusion.rate sd.diffusion.rate
##   <fct>               <dbl>             <dbl>
## 1 hOgg1               0.127             0.269
## 2 EndoV               0.645             0.977
## 3 mEndoV              0.701             0.936
```

```
if(!file.exists("Processed/localMSD_sim_data.rds")){

## calculation of average diffusion rate of Each proteins

localMSDs %>% filter(localMSD_15!=0, `Delta_X > 300` == "Yes", On_DNA)%>%
  group_by(Enzyme) %>%
  summarise(ave.diffusion.rate = mean(localMSD_15, na.rm = T),
            sd.diffusion.rate = sd(localMSD_15, na.rm = T))


#####

## simulation

##hOgg1

sim.hOgg1 <- NULL
for (i in 1:4000) {
  D = 0.13
  t = 0.0235
  SD <- sqrt(2 * D * t)
  x0 = 0
  n = sample(x = 5:100, size = 1)
  Step <- c(x0, rnorm(n = (n - 1), mean = 0, sd = SD))
  corrected_X <- cumsum(Step)
  Traj <- data.frame(Frame_number = seq(1, n, 1), corrected_X,
                     Step, Unique_trajectory_ID = 50000 + i)
  sim.hOgg1 <- rbind(sim.hOgg1, Traj)
  print(i)
}

sim.hOgg1 %<>% mutate(Enzyme = "hOgg1", Frame_interval = 23.5,
                      corrected_X = corrected_X*1000)
```

```
## Endov
sim.EndoV <- NULL
for (i in 1:4000) {
  D = 0.65
  t = 0.0075
  SD <- sqrt(2 * D * t)
  x0 = 0
  n = sample(x = 5:100, size = 1)
  Step <- c(x0, rnorm(n = (n - 1), mean = 0, sd = SD))
  corrected_X <- cumsum(Step)
  Traj <- data.frame(Frame_number = seq(1, n, 1), corrected_X,
                     Step, Unique_trajectory_ID = 55000 + i)
  sim.EndoV <- rbind(sim.EndoV, Traj)
  print(i)
}

sim.EndoV %<>% mutate(Enzyme = "EndoV", Frame_interval = 7.5,
                      corrected_X = corrected_X*1000)


## mEndoV

sim.mEndoV <- NULL
for (i in 1:4000) {
  D = 0.70
  t = 0.0075
  SD <- sqrt(2 * D * t)
  x0 = 0
  n = sample(x = 5:100, size = 1)
  Step <- c(x0, rnorm(n = (n - 1), mean = 0, sd = SD))
  corrected_X <- cumsum(Step)
  Traj <- data.frame(Frame_number = seq(1, n, 1), corrected_X,
                     Step, Unique_trajectory_ID = 60000 + i)
  sim.mEndoV <- rbind(sim.mEndoV, Traj)
  print(i)
}

sim.mEndoV %<>% mutate(Enzyme = "mEndoV", Frame_interval = 7.5,
                      corrected_X = corrected_X*1000)

sim.data <- bind_rows(sim.EndoV, sim.mEndoV, sim.hOgg1)

## localMSDs of simulations

localMSDs.sim <- sim.data %>%
  group_by(Unique_trajectory_ID) %>%
  mutate(time=Frame_number-min(Frame_number)) %>%
  mutate(localMSD_05 = localMSDcomplete(corrected_X, 5)/ (2000*Frame_interval),
         localMSD_07 = localMSDcomplete(corrected_X, 7)/ (2000*Frame_interval),
         localMSD_10 = localMSDcomplete(corrected_X, 10)/ (2000*Frame_interval),
         localMSD_15 = localMSDcomplete(corrected_X, 15)/ (2000*Frame_interval)) %>%
  ungroup()
```

```
localMSDs.sim$Enzyme <-  factor(localMSDs.sim$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))

## control

localMSDs.sim %>% filter(localMSD_15!=0) %>%
  group_by(Enzyme) %>%
  summarise(ave.diffusion.rate = mean(localMSD_15, na.rm = T),
            sd.diffusion.rate = sd(localMSD_15, na.rm = T))
saveRDS(localMSDs.sim, "Processed/localMSD_sim_data.rds")

}else{

localMSDs.sim <- readRDS("Processed/localMSD_sim_data.rds")
}

## plot sim and real data together
localMSDs$Enzyme <-  factor(localMSDs$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))


localMSDs.sim$Enzyme <-  factor(localMSDs.sim$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))

localMSDs %>%
  filter(localMSD_05!=0, `Delta_X > 300`== "Yes", On_DNA) %>%
  ggplot(aes(x=localMSD_05)) +
  facet_wrap(~Enzyme, ncol = 3, labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100, position = "stack",
                 aes(y=20*(..count..)/
                       tapply(..count..,..PANEL..,sum)[..PANEL..])) +
  geom_density(data =localMSDs.sim %>% filter(localMSD_05!=0) ,
               aes(x=localMSD_05),
               color = "black")+
  geom_density(color = "red")+
  scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Density")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2/s))) +
  scale_y_continuous(breaks = c(0.5, 1))+
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_fill_discrete(name = "Diffusion mode")
```

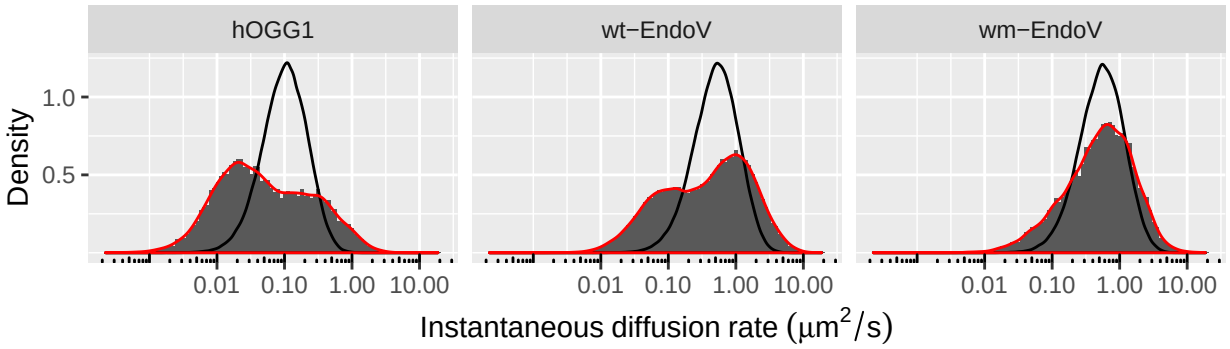

Figure 13: Instantaneous diffusion rate of the proteins and corresponding simulated single-mode random walks

7 Classification

7.1 Activation energy barrier classification

Assuming the scanning as a chemical reaction, the rate constant is $k = 1/t = 2D/ < x^2 >$ and from the Arrhenius equation we have $k = Ae^{(-E_a/K_BT)}$. In the case of ideal sliding the E_a is 0, therefore $A = k_i$. Plugging these into Arrhenius equation and solving for E_a, we have: $E_a = ln(k_i/k)k_BT$. Given the random walk has a 1 nucleotide stepping length, $< x^2 >$ is equal to $1bp^2$, therefore the ratio of k_i/k will be equal to D_i/D. Thus we get : $E_a = ln(D_i/D)k_BT$ where D_i is the upper theoretical limit of diffusion for helical sliding (calculated in section 9.3). By setting D as the instantaneous diffusion rate, we can calculate E_a for every step.

We classify the steps of trajectories based on the value of E_a into three categories, each representative of a particular scanning mode:

$E_a < 0.5k_BT$: hopping mode

$0.5k_BT < E_a < 2k_BT$: helical sliding mode

$E_a > 2k_BT$: interrogation mode

The result of this classification is shown in Fig.14.

```
localMSDs1 <- localMSDs %>%
  mutate(upper.limit = ifelse(Enzyme == "hOgg1", 0.89, 1.3)) %>%
  mutate(energy.barrier =
           ifelse(log(upper.limit/(localMSD_05)) <= 0.5 ,
                  "Ea < 0.5KbT",
                  ifelse(log(upper.limit/(localMSD_05)) >0.5 &
                           log(upper.limit/(localMSD_05)) <= 2,
                         "0.5KbT < Ea < 2KbT", "Ea > 2KbT"))) %>%
  filter(localMSD_05!=0, `Delta_X > 300`== "Yes", On_DNA)

localMSDs1$energy.barrier <-
  factor(localMSDs1$energy.barrier,
         levels = c('Ea > 2KbT','0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))

localMSDs1 %>%  ggplot(aes(x=localMSD_05)) +
```

```
  facet_wrap(~Enzyme, ncol = 3, labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100, position = "stack",
                 aes(y=20*(..count..)/tapply(..count..,..PANEL..,sum)[..PANEL..],
                     fill= energy.barrier )) +
  geom_density(color="red")+
  geom_density(data =localMSDs.sim %>% filter(localMSD_05!=0) ,
               aes(x=localMSD_05),
               color = "black")+
   scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Density")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2/s))) +
  scale_y_continuous(breaks = c(0.5, 1))+
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_fill_discrete(name = "Diffusion mode")
```

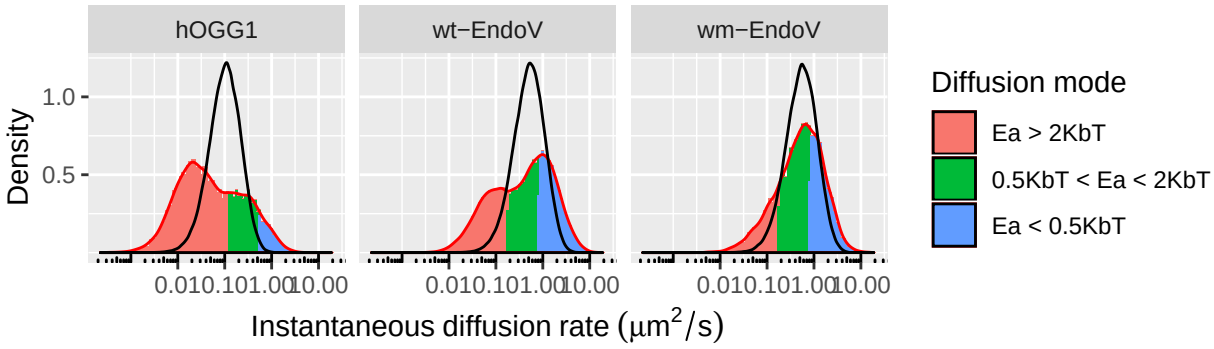

Figure 14: Energy barrier classification

7.2 Hidden Markov model classification

To verify the existence of the three above-mentioned modes of scanning, we used an analytical tool based on a variational Bayesian treatment of hidden Markov models (HMM)[4] to classify segments of the trajectories into three states. In the first step we need to transform the data to a version that is compatible with the MATLAB code of the vbSPT software for HMM classification. Using following scripts all trajectories are saved in separate text files using R and later they are read by MATLAB and concatenate as a cell array.

```
## hOgg1

localMSDs <- readRDS("Processed/localMSD_real_data.rds")

traj.id <- localMSDs %>%
  filter(Enzyme=="hOgg1", `Delta_X > 300` =="Yes") %>%
  extract2("Unique_trajectory_ID") %>%
  unique()

to.save2 <- localMSDs %>%
  filter(Enzyme=="hOgg1", `Delta_X > 300` =="Yes") %>%
  ungroup()
```

```
j=0

for(i in traj.id){
  j=j+1
  to.save <- to.save2 %>%
    filter(Unique_trajectory_ID == i) %>%
    select(corrected_X, Unique_trajectory_ID)

  write.table(to.save,
              paste("Processed/matlab/to_markov/hOgg1/" ,j,".txt",
                    sep = ""), sep="\t",col.names =F,row.names = F ,quote= F)
  print(j)

}

## EndoV

traj.id <- localMSDs %>%
  filter(Enzyme=="EndoV", `Delta_X > 300` =="Yes") %>%
  extract2("Unique_trajectory_ID") %>%
  unique()

to.save2 <- localMSDs %>%
  filter(Enzyme=="EndoV", `Delta_X > 300` =="Yes") %>%
  ungroup()

j=0

for(i in traj.id){
  j=j+1
  to.save <- to.save2 %>%
    filter(Unique_trajectory_ID == i) %>%
    select(corrected_X, Unique_trajectory_ID)
  write.table(to.save,
              paste("Processed/matlab/to_markov/EndoV/" ,j,".txt",
                    sep = ""), sep="\t",col.names =F,row.names = F ,quote= F)
  print(j)

}

## mEndoV

traj.id <- localMSDs %>%
  filter(Enzyme=="mEndoV", Delta_X =="Yes") %>%
  extract2("New_Unique_trajectory_ID") %>%
  unique()

to.save2 <- localMSDs %>%
  filter(Enzyme=="mEndoV", Delta_X =="Yes") %>%
  ungroup()

j=0
```

```
for(i in traj.id){
  j=j+1
  to.save <- to.save2 %>% filter(Unique_trajectory_ID == i) %>%
    select(corrected_X, Unique_trajectory_ID)
  write.table(to.save,
              paste("Processed/matlab/to_markov/mEndoV/" ,j,".txt",
                    sep = ""), sep="\t",col.names =F,row.names = F ,quote= F)
  print(j)

}
```

```
my_files = dir('*.txt');
for k = 1:length(my_files)
  textFileName = sprintf('%d.txt', k);
    textData = dlmread(textFileName);
    finalTraj{k} = textData;
      disp(k);
end
save hOgg1_data.mat finalTraj
```

The codes that are used in MATLAB for 3-state classification of the trajectories of the 3 proteins are as following. In order to perform the classification with higher number of states, the variable maxHidden must be increased relatively.

```
%% VB-HMM analysis parameter file generated by vbSPTgui %%

% Fredrik Persson & Martin Linden 2012-11-07

% This is a GUI generated HMM analysis runinput file, which specifies
% everything the code needs to know about how to analyze a particular data
% set.
% To run the HMM analysis manually type:
% >> VB3_HMManalysis('runinputfilename')


% Inputs
inputfile = '.\InputData\hOgg1_data.mat';
trajectoryfield = 'finalTraj';

% Computing strategy
parallelize_config = 1;
parallel_start = 'theVBpool=gcp';
% executed before the parallelizable loop.
parallel_end = 'delete(theVBpool)';
% executed after the parallelizable loop.

% Saving options
outputfile = '.\Results\hOgg1_3modes_final.mat';
jobID = 'Data from hOgg1_data.mat :: finalTraj from_22-Jun-2018.';

% Data properties
timestep = 0.0235;      % in [s]
dim = 1;
trjLmin = 5;
```

```
% Convergence and computation alternatives
runs = 25;
maxHidden = 3;

% Evaluate extra estimates including Viterbi paths
stateEstimate = 1;

maxIter = [];
% maximum number of VB iterations ([]: use default values).
relTolF = 1e-8;
% convergence criterion for relative change in likelihood bound.
tolPar = [];
% convergence criterion for M-step parameters (leave non-strict).

% Bootstrapping
bootstrapNum = 100;
fullBootstrap = 1;

% Limits for initial conditions
init_D = [0.005, 5]*1e6;
% interval for diffusion constant initial guess [length^2/time]
% in same length units as the input data.
init_tD = [5, 50]*timestep;
% interval for mean dwell time initial guess in [s].
% It is recommended to keep the initial tD guesses on the lower end of
% the expected spectrum.

% Prior distributions
% Diffusion constants
prior_type_D = 'mean_strength';
prior_D = 1e6;
% prior diffusion constant [length^2/time] in same length units as
% the input data.
prior_Dstrength = 5;
% strength of diffusion constant prior, number of pseudocounts (positive).

%% default prior choices (according nat. meth. 2013 paper)
prior_type_Pi = 'natmet13';
prior_piStrength = 5;
% prior strength of initial state distribution (assumed uniform)
% in pseudocounts.
prior_type_A = 'natmet13';
prior_tD = 10*timestep;
% prior dwell time in [s].
prior_tDstrength = 2*prior_tD/timestep;
% transition rate strength (number of pseudocounts). Recommended
% to be at least 2*prior_tD/timestep.

%% new prior choices (for advanced users, as they are not yet systematically tested)
%prior_type_Pi = 'flat';
%prior_type_A = 'dwell_Bflat';
%prior_tD = 10*timestep;
% prior dwell time in [s]. Must be greater than timestep
```

```
% (recommended > 2*timestep)
%prior_tDstd = 100*prior_tD;
% standard deviation of prior dwell times [s].
```

```
%% VB-HMM analysis parameter file generated by vbSPTgui %%

% Fredrik Persson & Martin Linden 2012-11-07

% This is a GUI generated HMM analysis runinput file, which specifies
% everything the code needs to know about how to analyze a particular data
% set.
% To run the HMM analysis manually type:
% >> VB3_HMManalysis('runinputfilename')


% Inputs
inputfile = '.\InputData\EndoV_data.mat';
trajectoryfield = 'finalTraj';

% Computing strategy
parallelize_config = 1;
parallel_start = 'theVBpool=gcp';
% executed before the parallelizable loop.
parallel_end = 'delete(theVBpool)';
% executed after the parallelizable loop.

% Saving options
outputfile = '.\Results\EndoV_3modes_final.mat';
jobID = 'Data from EndoV_data.mat :: finalTraj from_22-Jun-2018.';

% Data properties
timestep = 0.0075;      % in [s]
dim = 1;
trjLmin = 5;

% Convergence and computation alternatives
runs = 25;
maxHidden = 3;

% Evaluate extra estimates including Viterbi paths
stateEstimate = 1;

maxIter = [];
% maximum number of VB iterations ([]: use default values).
relTolF = 1e-8;
% convergence criterion for relative change in likelihood bound.
tolPar = [];
% convergence criterion for M-step parameters (leave non-strict).

% Bootstrapping
bootstrapNum = 100;
fullBootstrap = 1;

% Limits for initial conditions
```

```
init_D = [0.01, 10]*1e6;
% interval for diffusion constant initial guess [length^2/time] in
% same length units as the input data.
init_tD = [5, 50]*timestep;
% interval for mean dwell time initial guess in [s].
% It is recommended to keep the initial tD guesses on the lower end of
% the expected spectrum.

% Prior distributions
% Diffusion constants
prior_type_D = 'mean_strength';
prior_D = 1e6;
% prior diffusion constant [length^2/time] in same length
% units as the input data.
prior_Dstrength = 5;
% strength of diffusion constant prior, number of pseudocounts (positive).

%% default prior choices (according nat. meth. 2013 paper)
prior_type_Pi = 'natmet13';
prior_piStrength = 5;
% prior strength of initial state distribution (assumed uniform)
% in pseudocounts.
prior_type_A = 'natmet13';
prior_tD = 10*timestep;
% prior dwell time in [s].
prior_tDstrength = 2*prior_tD/timestep;
% transition rate strength (number of pseudocounts).
% Recommended to be at least 2*prior_tD/timestep.

%% new prior choices (for advanced users, as they are not yet
                    % systematically tested)
%prior_type_Pi = 'flat';
%prior_type_A = 'dwell_Bflat';
%prior_tD = 10*timestep;
% prior dwell time in [s]. Must be greater than timestep
% (recommended > 2*timestep)
%prior_tDstd = 100*prior_tD;
% standard deviation of prior dwell times [s].
```

```
%% VB-HMM analysis parameter file generated by vbSPTgui %%

% Fredrik Persson & Martin Linden 2012-11-07

% This is a GUI generated HMM analysis runinput file, which specifies
% everything the code needs to know about how to analyze a particular data
% set.
% To run the HMM analysis manually type:
% >> VB3_HMManalysis('runinputfilename')


% Inputs
inputfile = '.\InputData\mEndoV_data.mat';
trajectoryfield = 'finalTraj';
```

```
% Computing strategy
parallelize_config = 1;
parallel_start = 'theVBpool=gcp';
% executed before the parallelizable loop.
parallel_end = 'delete(theVBpool)';
% executed after the parallelizable loop.

% Saving options
outputfile = '.\Results\mEndoV_3modes_final.mat';
jobID = 'Data from mEndoV_data.mat :: finalTraj from_22-Jun-2018.';

% Data properties
timestep = 0.0075;     % in [s]
dim = 1;
trjLmin = 5;

% Convergence and computation alternatives
runs = 25;
maxHidden = 3;

% Evaluate extra estimates including Viterbi paths
stateEstimate = 1;

maxIter = [];
% maximum number of VB iterations ([]: use default values).
relTolF = 1e-8;
% convergence criterion for relative change in likelihood bound.
tolPar = [];
% convergence criterion for M-step parameters (leave non-strict).

% Bootstrapping
bootstrapNum = 100;
fullBootstrap = 1;

% Limits for initial conditions
init_D = [0.01, 10]*1e6;
% interval for diffusion constant initial guess [length^2/time]
% in same length units as the input data.
init_tD = [5, 50]*timestep;
% interval for mean dwell time initial guess in [s].
% It is recommended to keep the initial tD guesses on the lower end of
%the expected spectrum.

% Prior distributions
% Diffusion constants
prior_type_D = 'mean_strength';
prior_D = 1e6;
% prior diffusion constant [length^2/time] in same length
%units as the input data.
prior_Dstrength = 5;
% strength of diffusion constant prior, number of pseudocounts (positive).

%% default prior choices (according nat. meth. 2013 paper)
```

```
prior_type_Pi = 'natmet13';
prior_piStrength = 5;
% prior strength of initial state distribution
%(assumed uniform) in pseudocounts.
prior_type_A = 'natmet13';
prior_tD = 10*timestep;
% prior dwell time in [s].
prior_tDstrength = 2*prior_tD/timestep;
% transition rate strength (number of pseudocounts).
%Recommended to be at least 2*prior_tD/timestep.

%% new prior choices (for advanced users, as they are not yet systematically tested)
%prior_type_Pi = 'flat';
%prior_type_A = 'dwell_Bflat';
%prior_tD = 10*timestep;
% prior dwell time in [s].
%Must be greater than timestep (recommended > 2*timestep)
%prior_tDstd = 100*prior_tD;
% standard deviation of prior dwell times [s].
```

The vbSPT software selected more than three states to be the best fit for all proteins (8 states for wm- and wt-EndoV, 6 states for hOGG1). According to the publication describing the vbSPT software[4] the states with forbidden transitions, can acquire transition probabilities up to 0.012, therefore we assume any transition below this value as insignificant and assign 0 value for the transition probability. The transition probability matrices for all classification models including 3 or more number of states for all proteins are presented here. Using the functions of the "markovchain" package in R we checked whether these matrices are reducible or not.

For wt-EndoV:

```
transitions <- readRDS("Processed/transition_matrix.rds")

EndoVTPMs <- transitions[[1]]

transition.matrix <- new( "markovchain", transitionMatrix = EndoVTPMs[[1]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 3-state classification of wt-EndoV")
```

```
## [1] "Transition matrix of 3-state classification of wt-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3
## 1 0.957 0.029 0.014
## 2 0.032 0.931 0.037
## 3 0.014 0.053 0.934
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
```

```
## Closed classes:
## 1 2 3
## Recurrent classes:
## {1,2,3}
## Transient classes:
## NONE
## The Markov chain is irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = EndoVTPMs[[2]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 4-state classification of wt-EndoV")
```

```
## [1] "Transition matrix of 4-state classification of wt-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4
## 1 0.937 0.039 0.000 0.023
## 2 0.050 0.939 0.000 0.000
## 3 0.000 0.000 0.975 0.019
## 4 0.038 0.046 0.088 0.828
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 2 3 4
## Recurrent classes:
## {1,2,3,4}
## Transient classes:
## NONE
## The Markov chain is irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = EndoVTPMs[[3]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 5-state classification of wt-EndoV")
```

```
## [1] "Transition matrix of 5-state classification of wt-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5
## 1 0.954 0.000 0.037 0.000 0.000
## 2 0.000 0.777 0.017 0.000 0.206
## 3 0.047 0.000 0.944 0.000 0.000
## 4 0.000 0.000 0.000 0.984 0.000
```

```
## 5 0.021 0.051 0.038 0.084 0.807
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 3
## 4
## Recurrent classes:
## {1,3},{4}
## Transient classes:
## {2,5}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = EndoVTPMs[[4]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 6-state classification of wt-EndoV")
```

```
## [1] "Transition matrix of 6-state classification of wt-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5     6
## 1 0.953 0.000 0.036 0.000 0.000 0.000
## 2 0.000 0.813 0.000 0.044 0.000 0.143
## 3 0.047 0.000 0.946 0.000 0.000 0.000
## 4 0.000 0.014 0.000 0.973 0.000 0.000
## 5 0.000 0.000 0.000 0.043 0.954 0.000
## 6 0.053 0.119 0.111 0.055 0.000 0.663
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 3
## Recurrent classes:
## {1,3}
## Transient classes:
## {2,4,6},{5}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = EndoVTPMs[[5]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
```

```
print("Transition matrix of 7-state classification of wt-EndoV")
```

```
## [1] "Transition matrix of 7-state classification of wt-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5     6     7
## 1 0.955 0.000 0.031 0.000 0.000 0.000 0.014
## 2 0.000 0.808 0.000 0.012 0.028 0.000 0.152
## 3 0.021 0.000 0.944 0.035 0.000 0.000 0.000
## 4 0.000 0.019 0.052 0.922 0.000 0.000 0.000
## 5 0.000 0.000 0.000 0.000 0.984 0.000 0.000
## 6 0.000 0.000 0.000 0.000 0.040 0.954 0.000
## 7 0.042 0.127 0.000 0.141 0.014 0.016 0.648
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 5
## Recurrent classes:
## {5}
## Transient classes:
## {1,2,3,4,7},{6}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = EndoVTPMs[[6]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 8-state classification of wt-EndoV")
```

```
## [1] "Transition matrix of 8-state classification of wt-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5     6     7     8
## 1 0.940 0.000 0.000 0.046 0.000 0.000 0.000 0.014
## 2 0.000 0.804 0.000 0.000 0.000 0.029 0.000 0.155
## 3 0.000 0.000 0.935 0.000 0.065 0.000 0.000 0.000
## 4 0.111 0.000 0.000 0.889 0.000 0.000 0.000 0.000
## 5 0.000 0.016 0.079 0.000 0.897 0.000 0.000 0.000
## 6 0.000 0.000 0.000 0.000 0.000 0.989 0.000 0.000
## 7 0.000 0.000 0.000 0.000 0.000 0.033 0.959 0.000
## 8 0.047 0.127 0.000 0.000 0.154 0.000 0.023 0.641
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
```

```
## Closed classes:
## 6
## Recurrent classes:
## {6}
## Transient classes:
## {1,2,3,4,5,8},{7}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

For wm-EndoV:

```
mEndoVTPMs <- transitions[[2]]

transition.matrix <- new( "markovchain", transitionMatrix = mEndoVTPMs[[1]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 3-state classification of wm-EndoV")
```

```
## [1] "Transition matrix of 3-state classification of wm-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3
## 1 0.954 0.029 0.017
## 2 0.022 0.946 0.032
## 3 0.016 0.075 0.909
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 2 3
## Recurrent classes:
## {1,2,3}
## Transient classes:
## NONE
## The Markov chain is irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = mEndoVTPMs[[2]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 4-state classification of wm-EndoV")
```

```
## [1] "Transition matrix of 4-state classification of wm-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4
## 1 0.930 0.040 0.000 0.030
## 2 0.038 0.955 0.000 0.000
```

```
## 3 0.000 0.000 0.972 0.022
## 4 0.038 0.043 0.107 0.813
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 2 3 4
## Recurrent classes:
## {1,2,3,4}
## Transient classes:
## NONE
## The Markov chain is irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = mEndoVTPMs[[3]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 5-state classification of wm-EndoV")
```

```
## [1] "Transition matrix of 5-state classification of wm-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5
## 1 0.931 0.033 0.000 0.000 0.033
## 2 0.031 0.951 0.000 0.000 0.015
## 3 0.000 0.000 0.978 0.000 0.000
## 4 0.000 0.000 0.052 0.938 0.000
## 5 0.112 0.176 0.040 0.017 0.655
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 3
## Recurrent classes:
## {3}
## Transient classes:
## {1,2,5},{4}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = mEndoVTPMs[[4]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 6-state classification of wm-EndoV")
```

```
## [1] "Transition matrix of 6-state classification of wm-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5     6
## 1 0.954 0.000 0.037 0.000 0.000 0.000
## 2 0.000 0.799 0.000 0.037 0.000 0.164
## 3 0.029 0.000 0.958 0.000 0.000 0.000
## 4 0.000 0.000 0.000 0.975 0.013 0.000
## 5 0.000 0.000 0.000 0.056 0.941 0.000
## 6 0.047 0.115 0.132 0.049 0.000 0.658
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 3
## 4 5
## Recurrent classes:
## {1,3},{4,5}
## Transient classes:
## {2,6}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = mEndoVTPMs[[5]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 7-state classification of wm-EndoV")
```

```
## [1] "Transition matrix of 7-state classification of wm-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5     6     7
## 1 0.950 0.000 0.041 0.000 0.000 0.000 0.000
## 2 0.000 0.805 0.000 0.015 0.019 0.000 0.161
## 3 0.034 0.000 0.956 0.000 0.000 0.000 0.000
## 4 0.000 0.013 0.000 0.987 0.000 0.000 0.000
## 5 0.000 0.000 0.000 0.000 0.984 0.000 0.000
## 6 0.000 0.000 0.000 0.000 0.063 0.919 0.000
## 7 0.051 0.109 0.110 0.033 0.048 0.000 0.650
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 3
## 5
```

```
## Recurrent classes:
## {1,3},{5}
## Transient classes:
## {2,4,7},{6}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = mEndoVTPMs[[6]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 8-state classification of wm-EndoV")
```

```
## [1] "Transition matrix of 8-state classification of wm-EndoV"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5     6     7     8
## 1 0.941 0.000 0.041 0.000 0.000 0.015 0.000 0.000
## 2 0.000 0.801 0.000 0.000 0.025 0.000 0.000 0.164
## 3 0.039 0.000 0.938 0.000 0.000 0.020 0.000 0.000
## 4 0.000 0.000 0.000 0.985 0.000 0.000 0.000 0.000
## 5 0.000 0.000 0.000 0.000 0.981 0.000 0.000 0.000
## 6 0.088 0.000 0.100 0.000 0.000 0.811 0.000 0.000
## 7 0.000 0.000 0.000 0.000 0.056 0.000 0.936 0.000
## 8 0.038 0.122 0.029 0.109 0.021 0.000 0.037 0.633
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 3 6
## 4
## 5
## Recurrent classes:
## {1,3,6},{4},{5}
## Transient classes:
## {2,8},{7}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

For hOGG1:

```
hOGG1TPMs <- transitions[[3]]

transition.matrix <- new( "markovchain", transitionMatrix = hOGG1TPMs[[1]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 3-state classification of hOGG1")
```

```
## [1] "Transition matrix of 3-state classification of hOGG1"
```

```
print( transition.matrix )
```

```
##       1     2     3
## 1 0.967 0.032 0.000
## 2 0.050 0.925 0.025
## 3 0.032 0.172 0.795
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 2 3
## Recurrent classes:
## {1,2,3}
## Transient classes:
## NONE
## The Markov chain is irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = hOGG1TPMs[[2]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 4-state classification of hOGG1")
```

```
## [1] "Transition matrix of 4-state classification of hOGG1"
```

```
print( transition.matrix )
```

```
##       1     2     3     4
## 1 0.963 0.032 0.000 0.000
## 2 0.045 0.907 0.048 0.000
## 3 0.017 0.106 0.876 0.000
## 4 0.000 0.023 0.018 0.955
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 2 3
## Recurrent classes:
## {1,2,3}
## Transient classes:
## {4}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = hOGG1TPMs[[3]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
```

```
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 5-state classification of hOGG1")
```

```
## [1] "Transition matrix of 5-state classification of hOGG1"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5
## 1 0.955 0.034 0.000 0.000 0.000
## 2 0.049 0.936 0.000 0.000 0.000
## 3 0.000 0.022 0.914 0.064 0.000
## 4 0.047 0.053 0.157 0.742 0.000
## 5 0.000 0.000 0.012 0.000 0.971
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
## 1 2
## Recurrent classes:
## {1,2}
## Transient classes:
## {3,4},{5}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

```
transition.matrix <- new( "markovchain", transitionMatrix = hOGG1TPMs[[4]])
transition.matrix@transitionMatrix[transition.matrix@transitionMatrix<0.012] <-
  0
transition.matrix@transitionMatrix <- round(transition.matrix@transitionMatrix,
                                            digits = 3)
print("Transition matrix of 6-state classification of hOGG1")
```

```
## [1] "Transition matrix of 6-state classification of hOGG1"
```

```
print( transition.matrix )
```

```
##       1     2     3     4     5     6
## 1 0.956 0.034 0.000 0.000 0.000 0.000
## 2 0.055 0.936 0.000 0.000 0.000 0.000
## 3 0.000 0.000 0.778 0.000 0.221 0.000
## 4 0.000 0.026 0.015 0.953 0.000 0.000
## 5 0.043 0.047 0.064 0.103 0.741 0.000
## 6 0.000 0.000 0.000 0.000 0.013 0.972
```

```
print("The features of the transition matrix ")
```

```
## [1] "The features of the transition matrix "
```

```
summary(transition.matrix)
```

```
## Unnamed Markov chain  Markov chain that is composed by:
## Closed classes:
```

```
## 1 2
## Recurrent classes:
## {1,2}
## Transient classes:
## {3,4,5},{6}
## The Markov chain is not irreducible
## The absorbing states are: NONE
```

While most of the classification models with more than 3 states have reducible transition matrices, all the 3-state models have irreducible transition matrices. Next, the output files for 3-states classifications are transferred into R, and the information of the new classification is added to the rest of data. Here we first look at the distribution of those states in the instantaneous diffusion plot (Fig. 15).

```
## hOgg1

if(!file.exists("Processed/segmented_localMSD.rds"))
{
add.col<-function(df, Markov.state) {n.row<-dim(df)[1]
length(Markov.state)<-n.row
cbind(df, Markov.state)
}

data <- readMat('Processed/matlab/from_markov/hOgg1_3modes_final.mat')
a1 <- data[[2]][[10]][[5]]

traj.id <- localMSDs %>%
  filter(Enzyme=="hOgg1", `Delta_X > 300` =="Yes") %>%
  extract2("Unique_trajectory_ID") %>%
  unique()

to.save2 <- localMSDs %>%
  filter(Enzyme=="hOgg1", `Delta_X > 300` =="Yes") %>%
  ungroup()

j=0
to.out.hOgg1 <- NULL

for(i in traj.id){
  j=j+1
  to.save <- to.save2 %>%
    filter(Unique_trajectory_ID == i) %>%
    add.col(a1[[j]] %>% unlist())

  to.out.hOgg1 <- bind_rows(to.out.hOgg1, to.save)
  print(j)
  }

## Endov

data <- readMat('Processed/matlab/from_markov/EndoV_3modes_final.mat')
a1 <- data[[2]][[10]][[5]]

traj.id <- localMSDs %>%
  filter(Enzyme=="EndoV", `Delta_X > 300` =="Yes") %>%
  extract2("Unique_trajectory_ID") %>%
```

```
  unique()

to.save2 <- localMSDs %>%
  filter(Enzyme=="EndoV", `Delta_X > 300` =="Yes") %>%
  ungroup()

j=0
to.out.EndoV <- NULL

for(i in traj.id){
  j=j+1
  to.save <- to.save2 %>%
    filter(Unique_trajectory_ID == i) %>%
    add.col(a1[[j]] %>% unlist())

  to.out.EndoV <- bind_rows(to.out.EndoV, to.save)
  print(j)

}



## mEndov

data <- readMat('Processed/matlab/from_markov/mEndoV_3modes_final.mat')
a1 <- data[[2]][[10]][[5]]

traj.id <- localMSDs %>%
  filter(Enzyme=="mEndoV", `Delta_X > 300` =="Yes") %>%
  extract2("Unique_trajectory_ID") %>%
  unique()

to.save2 <- localMSDs %>%
  filter(Enzyme=="mEndoV", `Delta_X > 300` =="Yes") %>%
  ungroup()

j=0
to.out.mEndoV <- NULL

for(i in traj.id){
  j=j+1
  to.save <- to.save2 %>%
    filter(Unique_trajectory_ID == i) %>%
    add.col(a1[[j]] %>% unlist())

  to.out.mEndoV <- bind_rows(to.out.mEndoV, to.save)
  print(j)
  }


to.out <- bind_rows(to.out.hOgg1, to.out.EndoV, to.out.mEndoV)

segmented.localMSDs <- left_join(localMSDs %>%
```

```
                                  filter(`Delta_X > 300` == "Yes")%>%
  mutate(upper.limit = ifelse(Enzyme == "hOgg1", 0.89, 1.3)) %>%
  mutate(energy.barrier =
           ifelse(log(upper.limit/(localMSD_05)) <= 0.5 ,
                  "Ea < 0.5KbT",
                  ifelse(log(upper.limit/(localMSD_05)) >0.5 &
                           log(upper.limit/(localMSD_05)) <= 2,
                         "0.5KbT < Ea < 2KbT", "Ea > 2KbT"))),
  to.out) %>% arrange(Unique_trajectory_ID, Frame_number) %>%
  ungroup()
saveRDS(segmented.localMSDs, "Processed/segmented_localMSD.rds")
}else{
  segmented.localMSDs <-  readRDS("Processed/segmented_localMSD.rds")
}

segmented.localMSDs %>%
  filter(localMSD_05!=0, !is.na(Markov.state)) %>%
  ggplot(aes(x=localMSD_05)) +
  facet_wrap(~Enzyme, ncol = 3, labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100, position = "stack",
                 aes(y=20*(..count..)/tapply(..count..,..PANEL..,sum)[..PANEL..],
                     fill= as.factor(Markov.state))) +
  geom_density(color="red")+
  geom_density(data =localMSDs.sim %>% filter(localMSD_05!=0) ,
               aes(x=localMSD_05),
               color = "black")+
   scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Density")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2/s))) +
  scale_y_continuous(breaks = c(0.5, 1))+
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_fill_discrete(name = "Diffusion mode")
```

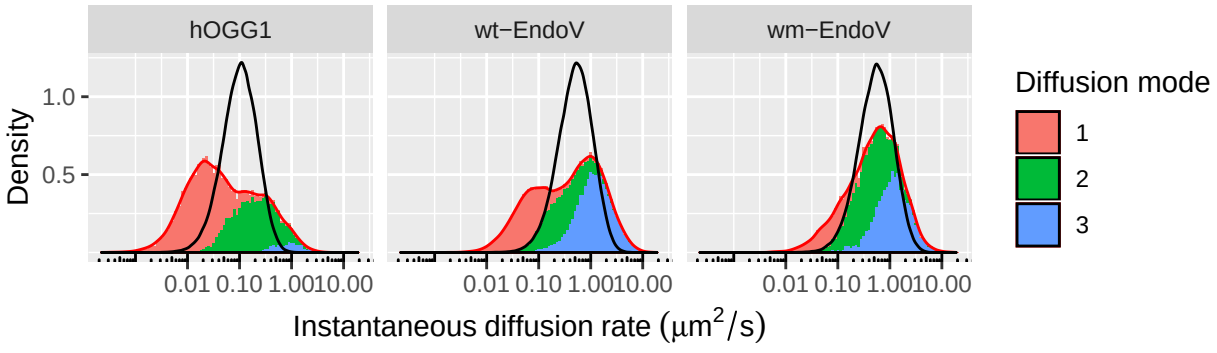

Figure 15: Hidden Markov model classification

To investigate the existence of non-Markovian transitions we checked the dynamic of mode switching for all conditions of three consecutive transitions for all proteins. There are 12 alternative sequences of three

consecutive transitions for each protein. We calculate the relative probabilities of the final transition given the same initial transition. The result of this analysis is shown in Fig. 16; the x axis shows the initial transition and the y axis is the relative probability of transition to either of two possible states given specific initial transitions.

```
temp <- segmented.localMSDs %>%
  arrange(New_Unique_trajectory_ID, Frame_number) %>%
  filter(!is.na(Markov.state)) %>%
  ungroup() %>%
  mutate(Markov.state0 = lag(Markov.state, n=1)) %>%
  group_by(New_Unique_trajectory_ID) %>%
  mutate(state.change = Markov.state- lag(Markov.state, n= 1)) %>%
  mutate(state.change = ifelse(is.na(state.change), 0, state.change)) %>%
  mutate(midd1 = abs(sign(state.change))) %>%
  mutate(transition = ifelse(midd1==1, Markov.state+ Markov.state0*10,0)) %>%
  filter(transition!=0) %>%
  mutate(transition0 = lag(transition, n=1)) %>%
  filter(!is.na(transition0)) %>%
  ungroup() %>%
  group_by(Enzyme, transition0, transition) %>%
  summarise( n = n()) %>%
  group_by(Enzyme, transition0) %>%
  mutate(total = sum(n), probability = n/total) %>%
  mutate(transition.seq = round(transition0/10)*100 + transition) %>%
  ungroup() %>%
  mutate(occurance.rate = n) %>%
  mutate(initial = round(transition0/10),
         middle = round(transition/10),
         final =  transition%%10)



temp$transition.seq <- factor(temp$transition.seq,
                              levels = c("121", "123", "321", "323",
                                         "212", "213", "312", "313",
                                         "131", "132", "231", "232"))
temp$transition0 <- factor(temp$transition0,
                              levels = c("12", "32", "21", "31", "13", "23"))

library(plyr)

temp$transition0 <- revalue(temp$transition0,
                            c("12"= "1>2", "32"= "3>2", "21"= "2>1",
                              "31"= "3>1", "13"= "1>3", "23"= "2>3"))

temp$Enzyme <-
  revalue(temp$Enzyme, c("hOgg1" = "hOGG1", "EndoV" = "wt-EndoV",
                                         "mEndoV" = "wm-EndoV"))

detach("package:plyr", unload=TRUE)


temp %>% arrange(Enzyme, transition.seq) %>%
   ggplot(aes(x =as.factor(transition0), y = probability,
```

```
            size = occurance.rate, color = as.factor(middle)))+
  geom_point(shape =17) +
  geom_text(aes(label=final),hjust=0.5, vjust=-1.5, size=3)+
  facet_wrap(~Enzyme) +
  xlab("Initail transition") +
  ylab("Relative probability of second transition\n given the same initail transition")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_y_continuous(limits = c(0,1.05))+
  scale_size_continuous(range = c(0.5, 5), breaks = c(50,100,200,300),
                        name = "Occurence rate")+
  scale_color_discrete(name = "Middle state")
```

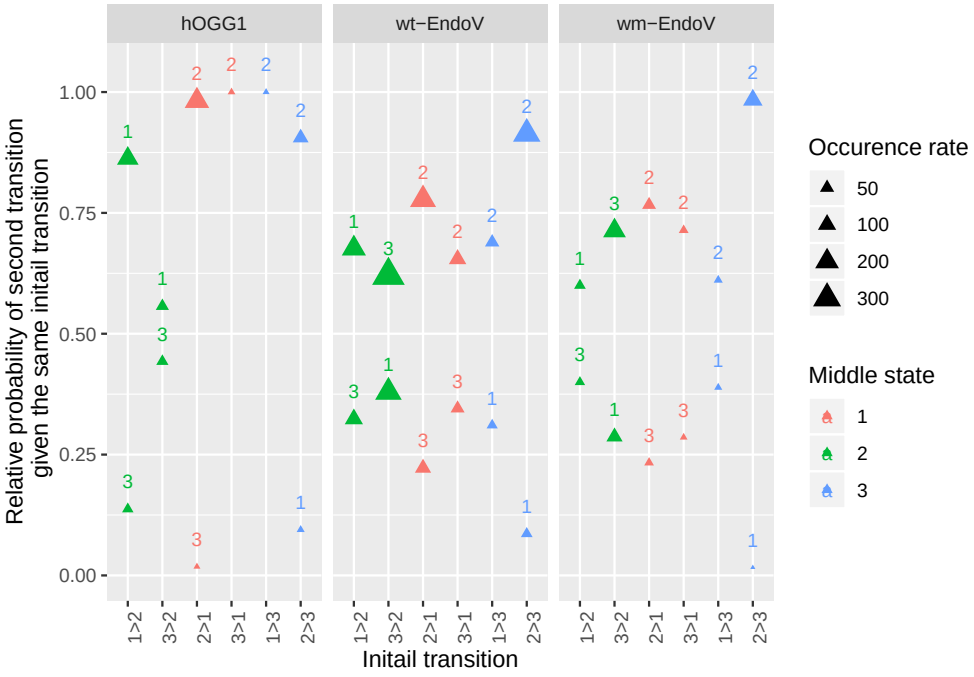

Figure 16: The effect of memory in sequence of transitions

In the next step, we want to check the consistency and overlap of the scanning modes from energy barrier-based classification and HMM-based classification. This is done first by looking into the confusion matrix that shows around 70% overlap between these two models.

```
localMSDs3 <- segmented.localMSDs %>% filter(localMSD_05!=0,
                                             !is.na(Markov.state))

library(plyr)

localMSDs3$energy.barrier <-
  revalue(localMSDs3$energy.barrier, c("Ea < 0.5KbT" = "3",
                                       "0.5KbT < Ea < 2KbT"= "2",
                                        "Ea > 2KbT"= "1"))

detach("package:plyr", unload=TRUE)
```

```
confusionMatrix(as.factor(localMSDs3$energy.barrier),
                as.factor(localMSDs3$Markov.state))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     1     2     3
##          1 28643  8107   835
##          2  2089 18523  9599
##          3   487  4435 23346
##
## Overall Statistics
##
##                Accuracy : 0.734
##                  95% CI : (0.7312, 0.7368)
##     No Information Rate : 0.3516
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6016
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.9175   0.5963   0.6911
## Specificity            0.8621   0.8202   0.9210
## Pos Pred Value         0.7621   0.6131   0.8259
## Neg Pred Value         0.9559   0.8095   0.8461
## Prevalence             0.3250   0.3234   0.3516
## Detection Rate         0.2982   0.1928   0.2430
## Detection Prevalence   0.3912   0.3145   0.2943
## Balanced Accuracy      0.8898   0.7082   0.8060
```

The agreement of the two models is further investigated by comparison of state occupancies and diffusion rates of the two models. In the energy barrier model, the frames are classified based on the value of the observed instantaneous diffusion rate. Since we use a moving window of 5 frames to calculate the instantaneous diffusion rate, the last four frames of each trajectory are not assigned any value for the instantaneous diffusion rate, and thus excluded from the energy barrier classification. Because of this, we excluded the same four frames of each trajectory after the HMM classification, and recalculated the diffusion rate of each state by averaging over the instantaneous diffusion rate of all remaining frames. The occupancy of each state was also recalculated as the proportion of time spent in each state. The recalculated values are close to the original values reported by the HMM without frame correction (Fig.17)

```
#####data from Markov #####

markov.data <- readRDS("Processed/markov_data.RDS")

markov.data$energy.barrier <-
  factor(markov.data$energy.barrier,
         levels = c('Ea > 2KbT',
                    '0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))

## Recalculation of Diffusion rates and occupancies of the Markov states
```

```
markov.result <- segmented.localMSDs %>%
  filter(localMSD_05!=0, !is.na(Markov.state)) %>%
  group_by(Enzyme, Markov.state) %>%
  summarise(diffusion.rate = mean(localMSD_05),
            diffusion.rate.error = sd(localMSD_05)/sqrt(n()),
            occupancy = n()) %>%
  group_by(Enzyme) %>%
  mutate(l= sum(occupancy)) %>%
  mutate(occupancy = occupancy/l) %>%
  select(-l) %>%
  mutate(Segmentation = "Markov states") %>%
  mutate(upper.limit = ifelse(Enzyme == "hOgg1", 0.89, 1.3)) %>%
  mutate(energy.barrier =
           ifelse(log(upper.limit/(diffusion.rate)) <= 0.5 ,
                  "Ea < 0.5KbT",
                  ifelse(log(upper.limit/(diffusion.rate)) > 0.5 &
                           log(upper.limit/(diffusion.rate)) <= 2,
                         "0.5KbT < Ea < 2KbT", "Ea > 2KbT"))) %>%
  select(-upper.limit)


energy.result <- segmented.localMSDs %>%
  filter(localMSD_05!=0, !is.na(Markov.state)) %>%
  group_by(Enzyme, energy.barrier) %>%
  summarise(diffusion.rate = mean(localMSD_05),
            diffusion.rate.error = sd(localMSD_05)/sqrt(n()),
                     occupancy = n()) %>%
  group_by(Enzyme) %>%
  mutate(l= sum(occupancy)) %>%
  mutate(occupancy = occupancy/l) %>% select(-l) %>%
  mutate(Segmentation = "Energy barrier")

diffusion.occupancy <- bind_rows( energy.result, markov.result)



library(plyr)

diffusion.occupancy$Enzyme <-
  revalue(diffusion.occupancy$Enzyme, c("hOgg1" = "hOGG1", "EndoV" = "wt-EndoV",
                                        "mEndoV" = "wm-EndoV"))

diffusion.occupancy$Segmentation <-
  revalue(diffusion.occupancy$Segmentation,
          c("Markov states" = "Recalculated values\n  for Markov States",
            "Energy barrier" = "Values for Energy\n  barrier states"))
detach("package:plyr", unload=TRUE)

combined.data <- bind_rows(diffusion.occupancy,
                markov.data %>%
                  filter(num.of.states == 3) %>%
                  select(-num.of.states, - upper.limit) %>%
                  mutate(Segmentation = "Original values\n  for Markov States"))
```

```
combined.data$energy.barrier <-
  factor(combined.data$energy.barrier,
         levels = c('Ea > 2KbT',
                    '0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))
combined.data %<>% arrange( Segmentation,Enzyme,  energy.barrier)

combined.data$Enzyme <-  factor(combined.data$Enzyme,
                                levels=c('hOGG1','wt-EndoV',
                                      'wm-EndoV'))

combined.data$Segmentation <-  factor(combined.data$Segmentation,
                                      levels=c('Values for Energy\n  barrier states',
                                         'Recalculated values\n  for Markov States',
                                         'Original values\n  for Markov States'))


theme_white4 <- theme(panel.background = element_blank(),
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.grid.major = element_line(color = "gray"),
                      axis.line = element_line(color = "black"),
                      axis.ticks.y = element_line(color = "black"),
                      axis.ticks.x = element_blank(),
                      strip.text = element_text (size=6, color= "black"),
                      axis.title.y = element_text(size = 6,
                                                  color = "black"),
                      axis.title.x = element_text(size = 6,
                                                  color = "black"),
                      axis.text.x = element_text (size=6, color= "black"),
                      axis.text.y = element_text (size=6, color= "black"),
                      legend.position="top",
                      legend.direction = "vertical",
                      legend.text = element_text(size=6),
                      legend.title = element_text(size=6),
                      panel.spacing = unit(10, "mm"))


combined.data %>% ggplot() +
  geom_point(aes(x = diffusion.rate,
                 y = occupancy,
                 shape= Segmentation,
                 color= energy.barrier), size =3, alpha = 0.8)+
  facet_wrap(~Enzyme,ncol=3, scales = "free_y")+
  theme_white4+
  ylab("Activation energy barrier range")+
  xlab(expression(Average~diffusion~rate~(mu*m^2/s))) +
  scale_shape_discrete(name = "Classification/diffusion\n calculatioin method")+
  facet_wrap(~Enzyme,ncol=3, scales = "free_y")+
  theme_white4+  scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Mode occupancy")+
```

```
  xlab(expression(Average~diffusion~rate~(mu*m^2/s))) +
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_color_discrete(name = "Energy barrier",
                       labels= c(expression(E[a]>~2*~k[B]*T),
                                 expression(0.5*~k[B]*T<~E[a]<~2*~k[B]*T),
                                 expression(E[a]<~0.5*~k[B]*T)))+
  scale_y_continuous(limits=c(0,0.8), breaks = c(0.25,0.5,0.75))
```

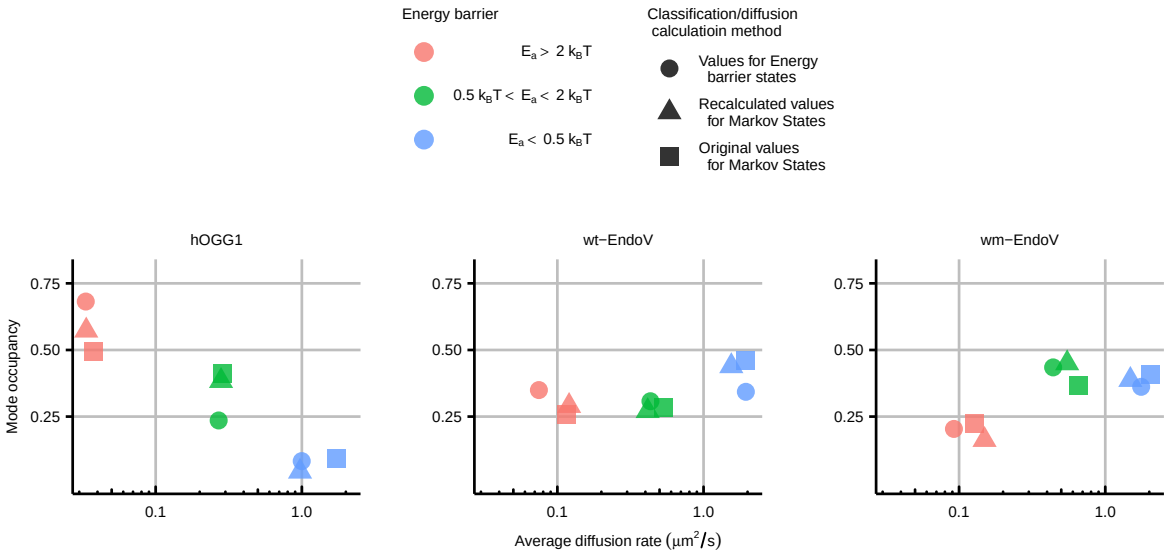

Figure 17: Hidden Markov model vs Energy barrier classification

8 Diffusion rate analysis of the classified data

Here we show the salt dependence of average diffusion rate of the diffusion modes based on both models of classifications (Fig. 18 and 19). The horizontal lines in the plots show the upper theoretical limit of diffusion rate.

```
segmented.localMSDs$energy.barrier <-
  factor(segmented.localMSDs$energy.barrier,
         levels = c('Ea > 2KbT','0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))

segmented.localMSDs %>%
  ungroup() %>%
  filter(localMSD_05!=0, Visual_confirmation=="Yes", NaCl>0) %>%
  group_by(Enzyme, NaCl, energy.barrier ) %>%
  summarise(meanMSD = mean(localMSD_05),
            errorMSD= sd(localMSD_05)/sqrt(n())) %>%
  ggplot(aes(x = NaCl, y = meanMSD, color = as.factor(energy.barrier)))+
  geom_point()+
```

```
  geom_errorbar(aes(x= NaCl,
                    ymax = meanMSD + errorMSD,
                    ymin = meanMSD - errorMSD,
                    color = as.factor(energy.barrier))) +
  facet_wrap(~Enzyme, labeller = as_labeller(labeloo3))+
  scale_x_log10()+
  geom_segment(data=data.frame(x=c(100,100,100),
                               y= c(0.89,1.30,1.30),
                               Enzyme=c("hOgg1","EndoV","mEndoV")),
               aes(x= 0, y= y, xend= x ,yend=y),
               inherit.aes=FALSE, size= 0.6)
```

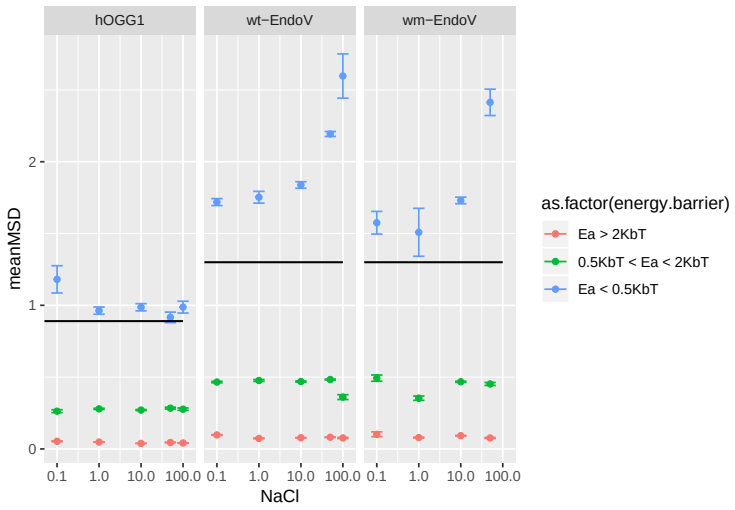

Figure 18: Salt dependenc of average diffusion rate

```
segmented.localMSDs %>%
  ungroup() %>% filter(localMSD_05!=0, NaCl>0, Visual_confirmation=="Yes") %>%
  group_by(Enzyme, NaCl, Markov.state ) %>%
  summarise(meanMSD = mean(localMSD_05),
            errorMSD= sd(localMSD_05)/sqrt(n())) %>%
  ggplot(aes(x = NaCl, y = meanMSD, color = as.factor(Markov.state)))+
  geom_point()+
  geom_errorbar(aes(x= NaCl,
                    ymax = meanMSD + errorMSD,
                    ymin = meanMSD - errorMSD,
                    color = as.factor(Markov.state)))+
  facet_wrap(~Enzyme, labeller = as_labeller(labeloo3))+
  scale_x_log10()+
  geom_segment(data=data.frame(x=c(100,100,100),
                               y= c(0.89,1.30,1.30),
                               Enzyme=c("hOgg1","EndoV","mEndoV")),
               aes(x= 0, y= y, xend= x ,yend=y),
               inherit.aes=FALSE, size= 0.6)
```

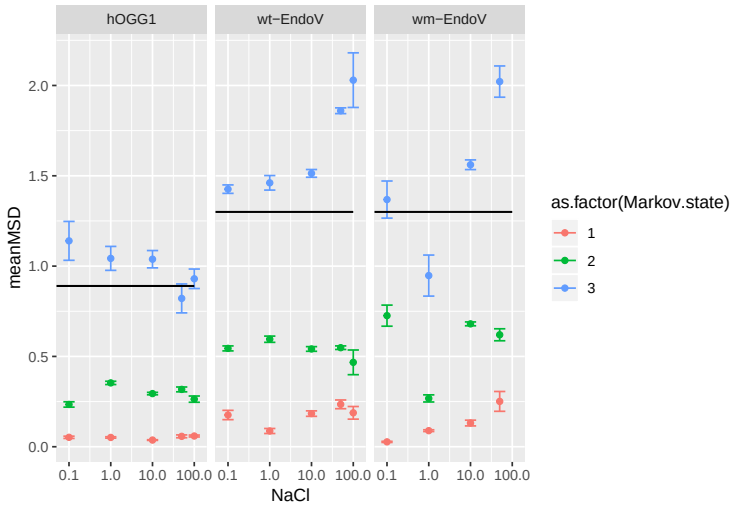

Figure 19: Salt dependenc of average diffusion rate

9 Numberical calculations used in the manuscript

9.1 Number of trajectories

Total number of trajectories calculated after applying three filters

1. being present for at least 5 consecutive frames
2. moving at least 300 nm along during the detection time
3. being within 200 nm spatial filters perpendicular to the detected DNA

```
aligned.data <-
  readRDS("Processed/2.1.AlignedDataBlinkingCorrected.2017-04-07.rds")

aligned.data%>%
  filter(On_DNA, `Delta_X > 300`== "Yes")%>%
  group_by(Enzyme) %>%
  summarize(trajectory_count=length(unique(New_Unique_trajectory_ID)),
            totalFrames=n())
```

```
## # A tibble: 3 x 3
##   Enzyme trajectory_count totalFrames
##   <chr>             <int>       <int>
## 1 EndoV              3443       70835
## 2 hOgg1               697       21309
## 3 mEndoV             1099       20081
```

9.2 Localization precision

From 118 proteins stock to the surface of the coverslips for at least 100 consecutive frames we calculate the localization precision as the standard deviation of detected signal and then average over all 118 data set.

```
aligned.data %>%
  filter(!On_DNA, !is.na(corrected_X), !is.na(corrected_Y)) %>%
```

```
  group_by(Enzyme, Unique_trajectory_ID) %>%
  summarise(sdX= sd(corrected_X), sdY= sd(corrected_Y), length=n()) %>%
  ungroup() %>%
  filter(length >100) %>%
  filter(!is.na(sdX)) %>%
  group_by(Enzyme) %>%
  summarise(precisionX = mean(sdX),
            precisionY = mean(sdY),
            lateral.precition= sqrt(precisionX^2 + precisionY^2))
```

```
## # A tibble: 3 x 4
##   Enzyme precisionX precisionY lateral.precition
##   <chr>       <dbl>      <dbl>             <dbl>
## 1 EndoV        30.6       28.0              41.5
## 2 hOgg1        26.2       25.6              36.6
## 3 mEndoV       31.9       29.8              43.6
```

```
aligned.data %>%
  filter(!On_DNA, !is.na(corrected_X), !is.na(corrected_Y)) %>%
  group_by(Enzyme, Unique_trajectory_ID) %>%
  summarise(sdX= sd(corrected_X), sdY= sd(corrected_Y), length=n()) %>%
  ungroup() %>%
  filter(length >100) %>%
  filter(!is.na(sdX)) %>%
  group_by(Enzyme) %>%
  summarise(precisionX = mean(sdX),
            SDprecisionX = sd(sdX),
            HighPrecisionX = round(precisionX- SDprecisionX),
            LowPrecisionX = round(precisionX+ SDprecisionX))
```

```
## # A tibble: 3 x 5
##   Enzyme precisionX SDprecisionX HighPrecisionX LowPrecisionX
##   <chr>       <dbl>        <dbl>          <dbl>         <dbl>
## 1 EndoV        30.6         7.95             23            39
## 2 hOgg1        26.2         6.52             20            33
## 3 mEndoV       31.9         9.82             22            42
```

```
# Now we want to know the average number of photons of the signal that have
#been detected from the moving proteins on DNA

 aligned.data %>%
   filter(Visual_confirmation== "Yes", On_DNA, !is.na(Intensity)) %>%
   group_by(Enzyme) %>%
  summarise(AveragePhotons = round(mean(Intensity)),
            SD= round(sd(Intensity)))
```

```
## # A tibble: 3 x 3
##   Enzyme AveragePhotons    SD
##   <chr>           <dbl> <dbl>
## 1 EndoV              95    48
## 2 hOgg1             109    64
## 3 mEndoV             87    45
```

9.3 Calculation of the upper limit of diffusion rate for helical sliding

Having the radius and the separation of center of DNA and protein for hOGG1 from literature, we can calculate the 3d diffusion rate of the proteins as well as upper limit of 1d diffusion rate of protein for helical sliding[5].

Here are the formulas to calculate the 3d diffusion and 1 rotational diffusion rate $D_{1d} = K_B T/6\eta\pi R$

$D_{3d} = K_B T/6\eta\pi R[1 + (2\pi/10BP)^2(4/3R^2 + R_{OC}^2)]$

We repeat the same calculations for EndoV, assuming that the molecular density of the two protein is the approximately similar we estimated the radius of EndoV from the ratio of their molecular weights.

```
# the values of hOGG1 from the literature

RochOGG1 = 2.5 * 1e-9
RhOGG1= 3.2 * 1e-9

# the ratio of radius from the ratio of molecular wieght
ratio <- (26/36)^(1/3)

RocEndoV = 2*1e-9
REndoV= RhOGG1 * ratio

#  Calculation of the diffusion rate

KB = 1.38*1e-23 # J/K: N.m/K
T = 296 #K
eta = 1.002*1e-3 # Ns/m2
BP= 0.34 *1e-9 * 0.95
# considering the fact that our DNA is stretched to 0.95 % of its contour length


## hOGG1
DhOGG13d <- KB*T/(6*pi*eta*RhOGG1)*1e12
DhOGG11d <- KB*T/((6*pi*eta*RhOGG1)*
                    (1+((2*pi/(10*BP))^2)*
                       (4*(RhOGG1^2)/3+RochOGG1^2)))*1e12

## EndoV

DEndoV3d <- KB*T/(6*pi*eta*REndoV)*1e12
DEndoV1d <- KB*T/((6*pi*eta*REndoV)*
                    (1+((2*pi/(10*BP))^2)*
                       (4*(REndoV^2)/3+RocEndoV^2)))*1e12

data.frame(Diffusion= c("3d diffusion rate of hOGG1: ",
                        "1d diffusion rate of hOGG1: ",
                        "3d diffusion rate of EndoV: ",
                        "1d diffusion rate of EndoV: " ),
                     Values =round (c(DhOGG13d, DhOGG11d,
                                      DEndoV3d, DEndoV1d),
                                    digits = 2))
```

```
##                       Diffusion Values
## 1 3d diffusion rate of hOGG1:   67.59
```

```
## 2 1d diffusion rate of hOGG1:    0.89
## 3 3d diffusion rate of EndoV:   75.33
## 4 1d diffusion rate of EndoV:    1.30
```

9.4 Switching mode calculation

In order to see the effect of wedge motif in the frequency of changing mode, we calculated that in average how many times each protein meets the confined mode per 1000 bp checked.

```
segmented.localMSDs %>%
  filter(localMSD_05!=0, `Delta_X > 300`== "Yes", On_DNA) %>%
  group_by(Unique_trajectory_ID) %>%
  mutate(length= n(), deltaBP= (max(corrected_X)- min(corrected_X))/
           (340*0.95)) %>%
  filter(length>5) %>%
  mutate(a= ifelse(Markov.state== lead(Markov.state, n=1),0,1)) %>%
  filter(a==1) %>% ungroup() %>%
  group_by(Unique_trajectory_ID) %>%
  mutate(b= ifelse(Markov.state=="1" |
                     lead(Markov.state, n=1)=="1",1,0),
         ModeChange= sum(b, na.rm= TRUE),
         ModeChPerBP= ModeChange/deltaBP) %>%
  ungroup() %>%
  select(Frame_number,Enzyme,Unique_trajectory_ID,
         length, Markov.state,a,b, ModeChange,
         deltaBP, ModeChPerBP) %>%
  group_by(Enzyme) %>%
  summarise(MeetTheStuckPer1kbp = mean(ModeChPerBP)/2)
```

```
## # A tibble: 3 x 2
##   Enzyme MeetTheStuckPer1kbp
##   <fct>                <dbl>
## 1 hOgg1                1.51
## 2 EndoV                0.677
## 3 mEndoV               0.269
```

9.5 Diffusion rate of scanning

Here we calculate

1. average diffusion rate of different proteins
2. average diffusion rate of different diffusion modes of the proteins based on energy barrier classification
3. average diffusion rate of different diffusion modes of the proteins based on Markov model classification

```
# Average diffusion rate of different proteins

segmented.localMSDs %>%
  filter(localMSD_15!=0, `Delta_X > 300` == "Yes", On_DNA)%>%
  group_by(Enzyme) %>%
  summarise(ave.diffusion.rate = mean(localMSD_15, na.rm = T),
            sd.diffusion.rate = sd(localMSD_15, na.rm = T))
```

```
## # A tibble: 3 x 3
```

```
##   Enzyme ave.diffusion.rate sd.diffusion.rate
##   <fct>               <dbl>             <dbl>
## 1 hOgg1               0.127             0.269
## 2 EndoV               0.645             0.977
## 3 mEndoV              0.701             0.936
```

```
# avrage diffusion rate of different diffusion modes of the proteins based on
# energy barrier classification

segmented.localMSDs %>%
  filter(localMSD_05!=0, `Delta_X > 300` == "Yes", On_DNA)%>%
  group_by(Enzyme, energy.barrier) %>%
  summarise(ave.diffusion.rate = mean(localMSD_05, na.rm = T),
            sd.diffusion.rate = sd(localMSD_15, na.rm = T))
```

```
## # A tibble: 9 x 4
## # Groups:   Enzyme [?]
##   Enzyme energy.barrier     ave.diffusion.rate sd.diffusion.rate
##   <fct>  <fct>                           <dbl>             <dbl>
## 1 hOgg1  Ea > 2KbT                      0.0330             0.113
## 2 hOgg1  0.5KbT < Ea < 2KbT             0.270              0.232
## 3 hOgg1  Ea < 0.5KbT                    0.994              0.558
## 4 EndoV  Ea > 2KbT                      0.0757             0.252
## 5 EndoV  0.5KbT < Ea < 2KbT             0.435              0.543
## 6 EndoV  Ea < 0.5KbT                    1.94               1.21
## 7 mEndoV Ea > 2KbT                      0.0928             0.214
## 8 mEndoV 0.5KbT < Ea < 2KbT             0.441              0.400
## 9 mEndoV Ea < 0.5KbT                    1.75               1.16
```

```
# avrage diffusion rate of different diffusion modes of the proteins based on
# Markov models classification

segmented.localMSDs %>%
  filter(localMSD_05!=0, `Delta_X > 300` == "Yes", On_DNA)%>%
  group_by(Enzyme, Markov.state) %>%
  summarise(ave.diffusion.rate = mean(localMSD_05, na.rm = T),
            sd.diffusion.rate = sd(localMSD_15, na.rm = T))
```

```
## # A tibble: 9 x 4
## # Groups:   Enzyme [?]
##   Enzyme Markov.state ave.diffusion.rate sd.diffusion.rate
##   <fct>         <int>              <dbl>             <dbl>
## 1 hOgg1             1             0.0331             0.112
## 2 hOgg1             2             0.283              0.279
## 3 hOgg1             3             0.963              0.555
## 4 EndoV             1             0.122              0.266
## 5 EndoV             2             0.419              0.414
## 6 EndoV             3             1.55               1.13
## 7 mEndoV            1             0.152              0.204
## 8 mEndoV            2             0.556              0.375
## 9 mEndoV            3             1.49               1.13
```

10 Presented figures

10.1 Figure 1

Fig. 1c

```
# Fig.1c --- Tracking output

LongDet <- readRDS("Processed/LongYes.rds")
LongRaw <- readRDS("Processed/LongRw.rds")

LongDet %<>%
  filter(x>3800, x<7000, y<1700)

theme_white1 <- theme(panel.background = element_blank(),
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.grid = element_blank(),
                      axis.line = element_blank(),
                      axis.ticks = element_blank(),
                      strip.text = element_blank(),
                      axis.title.y = element_blank(),
                      axis.title.x = element_blank(),
                      axis.text = element_blank())



LongDet %>%
  filter(r=="Yes") %>%
  ggplot(aes(x=x, y=y)) +
  geom_point(size=0.8, color='red', alpha=1)+
  geom_point(data=dplyr::anti_join(LongRaw, LongDet, by=c("x","y"))
             , color='black',alpha=0.1, size=0.1)+
  scale_x_continuous(limits = c(2340,8500)) +
  scale_y_continuous(limits = c(500,2628))+
  coord_equal(ratio=1) +
  theme_white1



# ggsave(filename = "Fig.1c3.svg" ,path= "Figures/" ,
#        width=14, height= 4.8, units= "cm", dpi=300)
```

10.2 Figure 2

Fig. 2a

```
# Fig.1a --- Instantaneous diffusion rate
segmented.localMSDs <-  readRDS("Processed/segmented_localMSD.rds")
localMSDs.sim <- readRDS("Processed/localMSD_sim_data.rds")
```

```
segmented.localMSDs$energy.barrier <-
  factor(segmented.localMSDs$energy.barrier,
         levels = c('Ea > 2KbT','0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))

segmented.localMSDs$Enzyme <-  factor(segmented.localMSDs$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))

localMSDs.sim$Enzyme <-  factor(localMSDs.sim$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))
labeloo3 <- c('EndoV'= "wt-EndoV" , 'hOgg1'= "hOGG1",
              'mEndoV'= "wm-EndoV")

theme_white2 <- theme(panel.background = element_blank(),
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.grid = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.ticks = element_line(color = "black"),
                      strip.text = element_text(size = 6, color = "black"),
                      axis.title.y = element_text(size = 6,color = "black"),
                      axis.title.x = element_text(size = 6,color = "black"),
                      axis.text = element_text(color = "black", size= 6),
                      legend.position="none",
                      legend.direction = "horizontal",
                      legend.text = element_text(size=6),
                      legend.title = element_text(size=6),
                      panel.spacing = unit(1, "mm"))

## plot



segmented.localMSDs %>%
  filter(localMSD_05!=0, `Delta_X > 300`== "Yes", On_DNA) %>%
  ggplot(aes(x=localMSD_05)) +
  facet_wrap(~Enzyme, ncol = 3, labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100, position = "stack",
                 aes(y=20*(..count..)/tapply(..count..,..PANEL..,sum)[..PANEL..],
                     fill= energy.barrier )) +
  geom_density(data =localMSDs.sim %>% filter(localMSD_05!=0) ,
               aes(x=localMSD_05),
               color = "black")+
   scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Density")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2/s))) +
  scale_y_continuous(breaks = c(0.5, 1))+
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
```

```
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_fill_discrete(name = "Diffusion mode")+
  theme_white2


# ggsave(filename = "Fig.2a.svg" ,path= "Figures/" ,
#        width=18.3, height= 4.8, units= "cm", dpi=300)
```

Fig 1.a (insets)

```
# Fig.1a, insets --- trajectories )

# select the trajectories

trajectories <-  aligned.data %>%
  filter(New_Unique_trajectory_ID %in% c(29580, 3621, 46999, 22334)) %>%
  group_by(Enzyme) %>%
  mutate(Time = (Frame_number- Frame_number[1])*Frame_interval/1000,
         Displacement = abs(corrected_X-min(corrected_X))/1000)

# theme

theme_white3 <- theme(panel.background = element_blank(),
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.border = element_rect (colour = "gray",
                                                   fill = F, size = 1),
                      panel.grid = element_blank(),
                      axis.line = element_line(color = "gray"),
                      axis.ticks = element_line(color = "gray"),
                      strip.text = element_blank(),
                      strip.placement= "inside",
                      axis.title.y = element_blank(),
                      axis.title.x = element_blank(),
                      axis.text = element_text(color = "black", size= 6),
                      legend.position = "none",
                      title = element_text(size=8, face="bold"))

# hOgg1

trajectories %>%
  filter(Enzyme== "hOgg1", Time> 0.5, Time < 3) %>%
  ggplot()+
  geom_line(aes(x= Time-Time[1], y= Displacement-min(Displacement)),
            size=0.1) +
  geom_point(aes(x= Time-Time[1], y= Displacement-min(Displacement)),
             size= 0.2, shape=16) +
  xlab("Time (s)")+
  ylab( expression(Position~(mu*m)))+
  scale_x_continuous(limits = c(0,2.5), breaks = c(0,1,2))+
  scale_y_continuous(limits = c(0,0.7), breaks = c(0,0.5))+
```

```
    theme_white3

# ggsave(filename = "Fig.2aInset1.svg" ,path= "Figures/"
#        , width=3, height= 2, units= "cm", dpi=300)

# EndoV

trajectories %>%
  filter(Enzyme== "EndoV") %>%
  ggplot()+
  geom_line(aes(x= Time, y= Displacement), size= 0.1) +
  geom_point(aes(x= Time, y= Displacement), size= 0.2, shape= 16)+
  xlab("Time (s)")+
  ylab( expression(Position~(mu*m)))+
  scale_x_continuous(limits = c(0,0.81), breaks = c(0,0.5))+
  scale_y_continuous(limits = c(0,1.11), breaks = c(0,0.5))+
  theme_white3

# ggsave(filename = "Fig.2aInset2.svg" ,path= "Figures/"
#        , width=3, height= 2, units= "cm", dpi=300)


# mEndoV

trajectories %>%
  filter(Enzyme== "mEndoV") %>%
  ggplot()+
  geom_line(aes(x= Time, y= Displacement), size= 0.1) +
  geom_point(aes(x= Time, y= Displacement), size= 0.2, shape= 16)+
  xlab("Time (s)")+
  ylab( expression(Position~(mu*m)))+
  scale_x_continuous(limits = c(0,0.61),
                     breaks = c(0,0.5))+
  scale_y_continuous(limits = c(0,1.75), breaks = c(0,1))+
    theme_white3

# ggsave(filename = "Fig.2aInset3.svg" ,path= "Figures/"
#        , width=3, height= 2, units= "cm", dpi=300)
```

Fig. 2b

```
# Fig.2b --- classification
diffusion.occupancy <- readRDS("Processed/diffusion_occupancy.rds")

diffusion.occupancy$Enzyme <-  factor(diffusion.occupancy$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))
diffusion.occupancy$energy.barrier <-
  factor(diffusion.occupancy$energy.barrier,
                                      levels = c('Ea > 2KbT',
                                                 '0.5KbT < Ea < 2KbT',
                                                 'Ea < 0.5KbT'))

theme_white4 <- theme(panel.background = element_blank(),
```

```
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.grid.major = element_line(color = "gray"),
                      axis.line = element_line(color = "black"),
                      axis.ticks.y = element_line(color = "black"),
                      axis.ticks.x = element_blank(),
                      strip.text = element_blank(),
                      axis.title.y = element_text(size = 6,
                                                  color = "black"),
                      axis.title.x = element_text(size = 6,
                                                  color = "black"),
                      axis.text.x = element_text (size=6, color= "black"),
                      axis.text.y = element_text (size=6, color= "black"),
                      legend.position="top",
                      legend.direction = "horizontal",
                      legend.text = element_text(size=6),
                      legend.title = element_text(size=6),
                      panel.spacing = unit(10, "mm"))

##plot

diffusion.occupancy %>% ggplot() +
  geom_point(aes(x = diffusion.rate,
                 y = occupancy,
                 color = energy.barrier ,
                 shape = Segmentation), size =3)+
  facet_wrap(~Enzyme)+
  facet_wrap(~Enzyme,ncol=3, scales = "free_y")+
  theme_white4+  scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Mode occupancy")+
  xlab(expression(Average~diffusion~rate~(mu*m^2/s))) +
   annotation_logticks(side= "b",
                       short = unit(0.3,"mm"),
                       mid = unit(0.6,"mm"),
                       long = unit(1,"mm"))+
  scale_color_discrete(name = "Energy barrier",
                       labels= c(expression(E[a]>~2*~k[B]*T),
                                 expression(0.5*~k[B]*T<~E[a]<~2*~k[B]*T),
                                 expression(E[a]<~0.5*~k[B]*T)))+
  scale_shape_discrete(name = "Classification method")+
  scale_y_continuous(limits=c(0,0.8), breaks = c(0.25,0.5,0.75))

# ggsave(filename = "Fig.2bb.svg" ,path= "Figures/" ,
#        width=18.3, height= 4, units= "cm", dpi=300)
```

10.3 Figure 3

```
# Fi.3a --- salt dependent of diffusion rate

segmented.localMSDs <- readRDS("Processed/segmented_localMSD.rds")
```

```
segmented.localMSDs$Enzyme <-  factor(segmented.localMSDs$Enzyme,
                                      levels=c('hOgg1','EndoV',
                                               'mEndoV'))
segmented.localMSDs$energy.barrier <-
  factor(segmented.localMSDs$energy.barrier,
         levels = c('Ea > 2KbT','0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))

theme_white6 <- theme(panel.background = element_blank(),
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.grid = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.ticks = element_line(color = "black"),
                      strip.text = element_text(size = 5,
                                                color = "black"),
                      axis.title.y = element_text(size = 6,
                                                  color = "black"),
                      axis.title.x = element_text(size = 6,
                                                  color = "black"),
                      axis.text = element_text(color = "black",
                                               size= 6),
                      legend.text = element_text(size=6),
                      legend.position = "top",
                      legend.title = element_text(size=6))

## plot

segmented.localMSDs %>% ungroup() %>% filter(localMSD_05!=0,
                                             Visual_confirmation=="Yes",
                                             NaCl>0) %>%
  group_by(Enzyme, NaCl, energy.barrier ) %>%
  summarise(meanMSD = mean(localMSD_05),
            errorMSD= sd(localMSD_05)/sqrt(n())) %>%
  ggplot(aes(x = NaCl, y = meanMSD, color = as.factor(energy.barrier)))+
  geom_point()+
  geom_errorbar(aes(x= NaCl,
                    ymax = meanMSD + errorMSD,
                    ymin = meanMSD - errorMSD,
                    color = as.factor(energy.barrier))) +
  scale_x_log10(breaks= c(0.1,1,10,100))+
  scale_y_continuous(limits=c(0,3), breaks = c(1,2))+
  theme_white6+
  facet_wrap(~Enzyme, scales = "free_y",
             labeller = as_labeller(labeloo3))+
  theme_white6+
  xlab("NaCl Concentration (mM)")+
  ylab(expression(Average~diffusion~rate~(mu*m^2~s^-1)))+
  scale_color_discrete(name = "Energy barrier",
                       labels= c(expression(E[a]>~2*~k[B]*T),
                                 expression(0.5*~k[B]*T<~E[a]<~2*~k[B]*T),
```

```
                                expression(E[a]<~0.5*~k[B]*T)))+
  annotation_logticks(side= "b",short = unit(0,"mm"),
                      mid = unit(0.6,"mm"), long = unit(1,"mm"))

#  ggsave(filename = "Fig.3a.svg" ,path= "Figures/" ,
#         width=8.8, height= 8, units= "cm", dpi=300)
#
# ggsave(filename = "Fig.3a.png" ,path= "Figures/" ,
#        width=8.9, height= 8, units= "cm", dpi=300)
#
# ggsave(filename = "Fig.3a.pdf" ,path= "Figures/" ,
#        width=8.9, height= 8, units= "cm", dpi=300)
```

10.4 Supplementary figure 1

```
# Fig.1a --- Instantaneous diffusion rate
segmented.localMSDs <-  readRDS("Processed/segmented_localMSD.rds")

segmented.localMSDs$Enzyme <-  factor(segmented.localMSDs$Enzyme,
                            levels=c('hOgg1','EndoV',
                                     'mEndoV'))

labeloo3 <- c('EndoV'= "wt-EndoV" , 'hOgg1'= "hOGG1",
              'mEndoV'= "wm-EndoV")

theme_white2 <- theme(panel.background = element_blank(),
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.grid = element_blank(),
                      axis.line = element_line(color = "black"),
                      axis.ticks = element_line(color = "black"),
                      strip.text = element_text(size = 6, color = "black"),
                      axis.title.y = element_text(size = 6,color = "black"),
                      axis.title.x = element_text(size = 6,color = "black"),
                      axis.text = element_text(color = "black", size= 6),
                      legend.position="none",
                      legend.direction = "horizontal",
                      legend.text = element_text(size=6),
                      legend.title = element_text(size=6),
                      panel.spacing = unit(1, "mm"),
                      plot.title = element_text(size = 8, color = "black"))

## plot

segmented.localMSDs %>%
  filter(localMSD_05!=0, `Delta_X > 300`== "Yes", On_DNA) %>%
  ggplot(aes(x=localMSD_05)) +
  facet_wrap(~Enzyme, ncol = 3, labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100, position = "stack",
                 aes(y=20*(..count..)/tapply(..count..,..PANEL..,sum)[..PANEL..])) +
```

```
  geom_density()+
   scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Density")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2~s^-1))) +
  scale_y_continuous(breaks = c(0.5, 1))+
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_fill_discrete(name = "Diffusion mode")+
  theme_white2+
  ggtitle("Moving window: 5 frames")


# ggsave(filename = "Fig.supp1a.png" ,path= "Figures/" ,
#         width=18, height= 4.6, units= "cm", dpi=300)



segmented.localMSDs %>%
  filter(localMSD_07!=0, `Delta_X > 300`== "Yes", On_DNA) %>%
  ggplot(aes(x=localMSD_07)) +
  facet_wrap(~Enzyme, ncol = 3, labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100, position = "stack",
                 aes(y=21*(..count..)/tapply(..count..,..PANEL..,sum)[..PANEL..])) +
  geom_density()+
   scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Density")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2~s^-1))) +
  scale_y_continuous(breaks = c(0.5, 1))+
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_fill_discrete(name = "Diffusion mode")+
  theme_white2+
    ggtitle("Moving window: 7 frames")


# ggsave(filename = "Fig.supp1b.png" ,path= "Figures/" ,
#         width=18, height= 4.6, units= "cm", dpi=300)


segmented.localMSDs %>%
  filter(localMSD_10!=0, `Delta_X > 300`== "Yes", On_DNA) %>%
  ggplot(aes(x=localMSD_10)) +
  facet_wrap(~Enzyme, ncol = 3, labeller = as_labeller(labeloo3)) +
  geom_histogram(bins=100, position = "stack",
                 aes(y=21*(..count..)/tapply(..count..,..PANEL..,sum)[..PANEL..])) +
  geom_density()+
   scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Density")+
  xlab( expression(Instantaneous~diffusion~rate~(mu*m^2~s^-1))) +
```

```
  scale_y_continuous(breaks = c(0.5, 1))+
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_fill_discrete(name = "Diffusion mode")+
  theme_white2+
  ggtitle("Moving window: 10 frames")


# ggsave(filename = "Fig.supp1c.png" ,path= "Figures/" ,
#         width=18, height= 4.6, units= "cm", dpi=300)
#
```

10.5 Supplementary figure 2

```
markov.data <- readRDS("Processed/markov_data.RDS")

markov.data$energy.barrier <-
  factor(markov.data$energy.barrier,
         levels = c('Ea > 2KbT',
                    '0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))

## Recalculation of Diffusion rates and occupancies of the Markov states

markov.result <- segmented.localMSDs %>%
  filter(localMSD_05!=0, !is.na(Markov.state)) %>%
  group_by(Enzyme, Markov.state) %>%
  summarise(diffusion.rate = mean(localMSD_05),
            diffusion.rate.error = sd(localMSD_05)/sqrt(n()),
            occupancy = n()) %>%
  group_by(Enzyme) %>%
  mutate(l= sum(occupancy)) %>%
  mutate(occupancy = occupancy/l) %>%
  select(-l) %>%
  mutate(Segmentation = "Markov states") %>%
  mutate(upper.limit = ifelse(Enzyme == "hOgg1", 0.89, 1.3)) %>%
  mutate(energy.barrier =
           ifelse(log(upper.limit/(diffusion.rate)) <= 0.5 ,
                  "Ea < 0.5KbT",
                  ifelse(log(upper.limit/(diffusion.rate)) > 0.5 &
                           log(upper.limit/(diffusion.rate)) <= 2,
                         "0.5KbT < Ea < 2KbT", "Ea > 2KbT"))) %>%
  select(-upper.limit)


energy.result <- segmented.localMSDs %>%
  filter(localMSD_05!=0, !is.na(Markov.state)) %>%
  group_by(Enzyme, energy.barrier) %>%
  summarise(diffusion.rate = mean(localMSD_05),
            diffusion.rate.error = sd(localMSD_05)/sqrt(n()),
```

```
                      occupancy = n()) %>%
  group_by(Enzyme) %>%
  mutate(l= sum(occupancy)) %>%
  mutate(occupancy = occupancy/l) %>% select(-l) %>%
  mutate(Segmentation = "Energy barrier")

diffusion.occupancy <- bind_rows( energy.result, markov.result)



library(plyr)

diffusion.occupancy$Enzyme <-
  revalue(diffusion.occupancy$Enzyme, c("hOgg1" = "hOGG1", "EndoV" = "wt-EndoV",
                                        "mEndoV" = "wm-EndoV"))

diffusion.occupancy$Segmentation <-
  revalue(diffusion.occupancy$Segmentation,
          c("Markov states" = "Recalculated values\n  for Markov States",
            "Energy barrier" = "Values for Energy\n  barrier states"))
detach("package:plyr", unload=TRUE)

combined.data <- bind_rows(diffusion.occupancy,
               markov.data %>%
                 filter(num.of.states == 3) %>%
                 select(-num.of.states, - upper.limit) %>%
                 mutate(Segmentation = "Original values\n  for Markov States"))
combined.data$energy.barrier <-
  factor(combined.data$energy.barrier,
         levels = c('Ea > 2KbT',
                    '0.5KbT < Ea < 2KbT',
                    'Ea < 0.5KbT'))
combined.data %<>% arrange( Segmentation,Enzyme,  energy.barrier)

combined.data$Enzyme <-  factor(combined.data$Enzyme,
                                levels=c('hOGG1','wt-EndoV',
                                      'wm-EndoV'))

combined.data$Segmentation <-  factor(combined.data$Segmentation,
                                      levels=c('Values for Energy\n  barrier states',
                                         'Recalculated values\n  for Markov States',
                                         'Original values\n  for Markov States'))


theme_white4 <- theme(panel.background = element_blank(),
                      legend.key = element_blank(),
                      legend.background = element_blank(),
                      strip.background = element_blank(),
                      plot.background = element_blank(),
                      panel.grid.major = element_line(color = "gray"),
                      axis.line = element_line(color = "black"),
                      axis.ticks.y = element_line(color = "black"),
                      axis.ticks.x = element_blank(),
```

```
                      strip.text = element_text (size=6, color= "black"),
                      axis.title.y = element_text(size = 6,
                                                  color = "black"),
                      axis.title.x = element_text(size = 6,
                                                  color = "black"),
                      axis.text.x = element_text (size=6, color= "black"),
                      axis.text.y = element_text (size=6, color= "black"),
                      legend.position="top",
                      legend.direction = "vertical",
                      legend.text = element_text(size=6),
                      legend.title = element_text(size=6),
                      panel.spacing = unit(10, "mm"))


combined.data %>% ggplot() +
  geom_point(aes(x = diffusion.rate,
                 y = occupancy,
                 shape= Segmentation,
                 color= energy.barrier), size =3, alpha = 0.8)+
  facet_wrap(~Enzyme,ncol=3, scales = "free_y")+
  theme_white4+
  ylab("Activation energy barrier range")+
  xlab(expression(Average~diffusion~rate~(mu*m^2/s))) +
  scale_shape_discrete(name = "Classification/diffusion\n calculatioin method")+
  facet_wrap(~Enzyme,ncol=3, scales = "free_y")+
  theme_white4+  scale_x_log10(breaks= c(0.01,0.1,1,10))+
  ylab("Mode occupancy")+
  xlab(expression(Average~diffusion~rate~(mu*m^2/s))) +
  annotation_logticks(side= "b",
                      short = unit(0.3,"mm"),
                      mid = unit(0.6,"mm"),
                      long = unit(1,"mm"))+
  scale_color_discrete(name = "Energy barrier",
                       labels= c(expression(E[a]>~2*~k[B]*T),
                                 expression(0.5*~k[B]*T<~E[a]<~2*~k[B]*T),
                                 expression(E[a]<~0.5*~k[B]*T)))+
  scale_y_continuous(limits=c(0,0.8), breaks = c(0.25,0.5,0.75))

# ggsave(filename = "Fig.supp2.png" ,path= "Figures/" ,
#         width=18, height= 7.5, units= "cm", dpi=300)
```

10.6 Supplementary figure 3

```
temp <- segmented.localMSDs %>%
  arrange(New_Unique_trajectory_ID, Frame_number) %>%
  filter(!is.na(Markov.state)) %>%
  ungroup() %>%
  mutate(Markov.state0 = lag(Markov.state, n=1)) %>%
  group_by(New_Unique_trajectory_ID) %>%
  mutate(state.change = Markov.state- lag(Markov.state, n= 1)) %>%
  mutate(state.change = ifelse(is.na(state.change), 0, state.change)) %>%
  mutate(midd1 = abs(sign(state.change))) %>%
```

```
  mutate(transition = ifelse(midd1==1, Markov.state+ Markov.state0*10,0)) %>%
  filter(transition!=0) %>%
  mutate(transition0 = lag(transition, n=1)) %>%
  filter(!is.na(transition0)) %>%
  ungroup() %>%
  group_by(Enzyme, transition0, transition) %>%
  summarise( n = n()) %>%
  group_by(Enzyme, transition0) %>%
  mutate(total = sum(n), probability = n/total) %>%
  mutate(transition.seq = round(transition0/10)*100 + transition) %>%
  ungroup() %>%
  mutate(occurance.rate = n) %>%
  mutate(initial = round(transition0/10),
         middle = round(transition/10),
         final =  transition%%10)



temp$transition.seq <- factor(temp$transition.seq,
                              levels = c("121", "123", "321", "323",
                                         "212", "213", "312", "313",
                                         "131", "132", "231", "232"))
temp$transition0 <- factor(temp$transition0,
                              levels = c("12", "32", "21", "31", "13", "23"))

library(plyr)

temp$transition0 <- revalue(temp$transition0,
                            c("12"= "1>2", "32"= "3>2", "21"= "2>1",
                              "31"= "3>1", "13"= "1>3", "23"= "2>3"))

temp$Enzyme <-
  revalue(temp$Enzyme, c("hOgg1" = "hOGG1", "EndoV" = "wt-EndoV",
                                         "mEndoV" = "wm-EndoV"))

detach("package:plyr", unload=TRUE)


theme_supp <- theme(  axis.line =  element_line(color = "black"),
                      axis.ticks = element_line(color = "black"),
                      strip.text = element_text(size = 6, color = "black"),
                      axis.title.y = element_text(size = 6,color = "black"),
                      axis.title.x = element_text(size = 6,color = "black"),
                      axis.text = element_text(color = "black", size= 6),
                      legend.position="right",
                      legend.direction = "vertical",
                      legend.text = element_text(size=6),
                      legend.title = element_text(size=6),
                      panel.spacing = unit(1, "mm"),
                      plot.title = element_text(size = 8, color = "black"))

temp %>% arrange(Enzyme, transition.seq) %>%
   ggplot(aes(x =as.factor(transition0), y = probability,
```

```
              size = occurance.rate, color = as.factor(middle)))+
  geom_point(shape =17) +
  geom_text(aes(label=final),hjust=0.5, vjust=-1.5, size=3)+
  facet_wrap(~Enzyme) +
  xlab("Initail transition") +
  ylab("Relative probability of second transition\n given the same initail transition")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_y_continuous(limits = c(0,1.05))+
  scale_size_continuous(range = c(0.5, 5), breaks = c(50,100,200,300),
                        name = "Occurence rate")+
  scale_color_discrete(name = "Middle state")+
  theme_supp
#
# ggsave(filename = "Fig.supp3.png" ,path= "Figures/" ,
#         width=18, height= 10, units= "cm", dpi=300)
#
#
```

11 References

1. Schindelin, J. et al. Fiji: An open-source platform for biological-image analysis. Nature Methods 9, 676-682 (2012).
2. Ovesný, M., Křížek, P., Borkovec, J., Svindrych, Z. & Hagen, G. M. ThunderSTORM: A comprehensive ImageJ plug-in for PALM and STORM data analysis and super-resolution imaging. Bioinformatics 30, 2389-2390 (2014).
3. Tinevez, J. Y. et al. TrackMate: An open and extensible platform for single-particle tracking. Methods 115, 80-90 (2017).
4. Persson, F., Lindén, M., Unoson, C. & Elf, J. Extracting intracellular diffusive states and transition rates from single-molecule tracking data. Nat. Methods 10, 265-9 (2013).
5. Bagchi, B., Blainey, P. C. & Sunney Xie, X. Diffusion constant of a nonspecifically bound protein undergoing curvilinear motion along DNA. J. Phys. Chem. B 112, 6282-6284 (2008).