Erklearung

Moritz Mennenga

17. Juni 2017

Hier eine kleine Erklärung zur Funktionsweise:

Ziel

Ziel ist es die Fotogrammetriepunkte eines Profils, die mit x, y und z Koordinaten beschrieben sind so zu rotieren und zu klappen, dass man im GIS in der 2D Aufsicht diese georeferenzieren kann.

Im Georeferencer gibt es den Transformationstyp projektiv, der eine Entzerrung ermöglicht.

Sind die Entzerrungen nicht an ihrer 3D-Position nötig, würde das für die meisten Fälle ausreichen.

Während bei PhoToPlan das BKS über die Punkte gelegt wird, werden hier die Punkte so rotiert, dass die Ansicht im GIS zum BKS wird.

Durchführung

Laden der Daten und auslesen nur eines Profils

Eingabe ist dabei ein SpatialDataFrame (shape)

```
devtools::load_all()
```

```
## Loading recexcavAAR
```

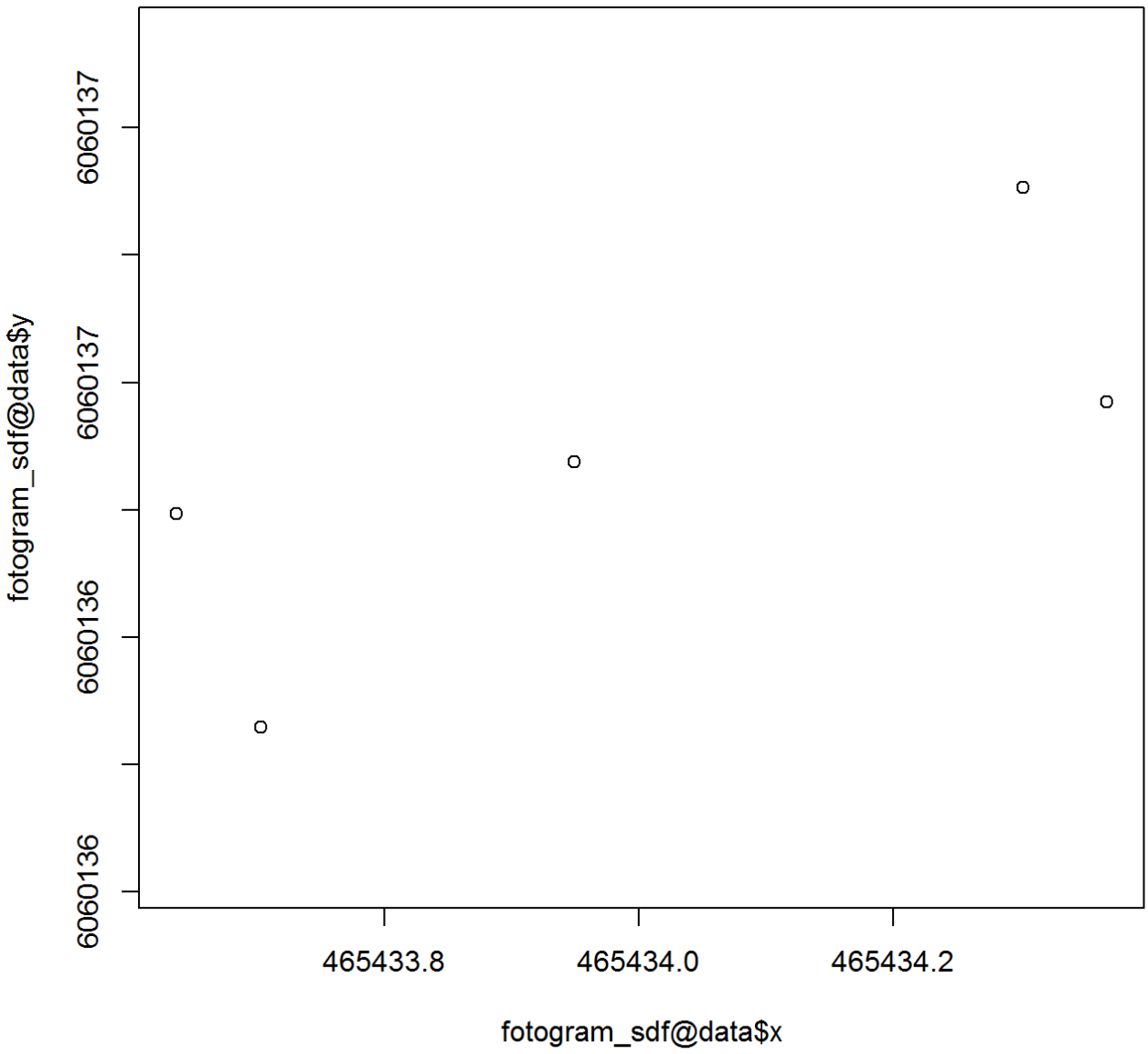
```
## Loading required package: kriging
```

```
## Warning: package 'kriging' was built under R version 3.3.3
```

```
library(sp)
fotogram_sdf <- sp::SpatialPointsDataFrame(coords = fotogram_pts[ ,c(1,2,3)],
data <- fotogram_pts,
proj4string = sp::CRS('+proj=utm +zone=32 +ellps=GRS80 +units=m +no_defs'))

#Fuer das Beispiel nur ein Profil
fotogram_sdf <- fotogram_sdf[fotogram_sdf@data$pr == 3,]
```

In der Ansicht im GIS würden die Messpunkte zunächst so aussehen. Das Profil ist in diesem Fall um 20° nach hinten gekippt, d.h. die Punkte in Süd-Ost liegen weiter unten.

```
plot(fotogram_sdf@data$x,fotogram_sdf@data$y,asp=1)
```



Um die Daten in den 2D Raum zu bringen und nach vorne zu klappen, müssen sie zunächst gedreht werden. Das ist später auch praktischer, wenn die Daten für den Druck exportiert werden sollen. *Hier könnte überlegt werden, ob man den Schritt auch optional macht, d.h. die Profile nachher wieder zurückdreht, so würde es die Möglichkeit geben, dass die Profile später auf der Schnittline des Befundes liegen*

Zwei Parameter sind wichtig, zum einen die Profilnummer, um die Profile einzeln zu bearbeiten und die Ansicht auf das Profil, denn diese bestimmt, in welche Richtung das Profil gedreht werden muss.

Da die Spalten nicht immer gleich sind - wohingegen die Koordinatenspalten in einem SpatialDataFrame fest definiert sind - können die Beschriftungen als Parameter übergeben werden und die Spalten werden herausgesucht.

```
fotogram_pts <- NULL
fotogram_pts <- fotogram_sdf

profile_col <- "pr" #Normalerweise bei Funktionsaufruf übergeben
view_col <- "view" #Normalerweise bei Funktionsaufruf übergeben

  profile_id <- 1
  while(profile_id<length(colnames(fotogram_pts@data)) &&
        colnames(fotogram_pts@data)[profile_id] != profile_col)
  {
    profile_id <- profile_id+1
  }

  #Spalte mit dem NAmen raussuchen und den Spaltenindex merken
  view_id <- 1
  while(view_id<length(colnames(fotogram_pts@data)) &&
        colnames(fotogram_pts@data)[view_id] != view_col)
  {
    view_id <- view_id+1
  }
fotogram_pts@data
```

```
##          x       y       z view pr
## 1 465433.7 6060136 4.36277    N  3
## 2 465434.4 6060137 4.36426    N  3
## 3 465433.9 6060137 4.65011    N  3
## 4 465433.6 6060136 4.83866    N  3
## 5 465434.3 6060137 4.84038    N  3
```

```
print(view_id)
```

```
## [1] 4
```

```
print(profile_id)
```

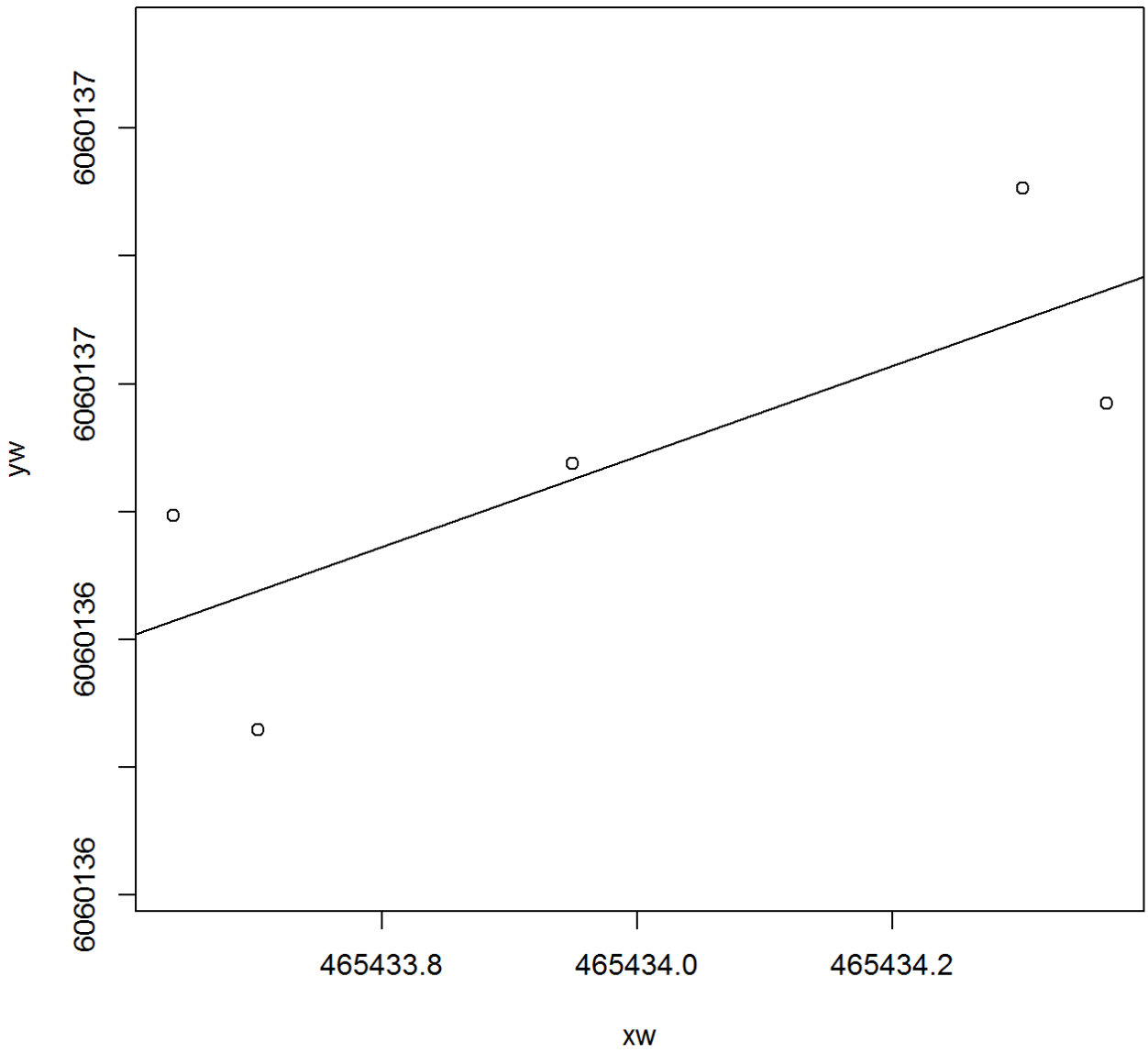
```
## [1] 5
```

Nun werden die Daten vorbereitet

```
  coord <- data.frame(fotogram_pts@coords[ ,1],
                      fotogram_pts@coords[ ,2],
                      fotogram_pts@coords[ ,3],
                      fotogram_pts@data[ ,profile_id],
                      fotogram_pts@data[, view_id])
  colnames(coord) <- c("x", "y", "z", "pr","view")


  #Jetzt muss jedes Profil einzeln bearbeitet werden
  #Nas raus
  coord <- na.omit(coord)
  #Alle Namen der Profile ermitteln, da sich daraus die
  #Anzahl der Schleifenduchläufe bestimmt
  prnames <- levels(as.factor(coord$pr))
  #Eine Tabelle, die für den Export da ist,
  #mit der gleichen Datenlänge wie die importierte
  #Da dieser df am ende immer wieder neue daten angehängt bekommt
  #und das rbind mit df blöd ist, wird hier eine Kopie der orginaltabelle überschr
ieben
  #damit fehler auffallen, sind alle Spalten = 0; Auch wenn es sehr unelegant ist
  coord_export <- coord
  coord_export$view <- NULL
  coord_export$pr <- 0
  coord_export$x <- 0
  coord_export$y <- 0
```

Für jedes Profil werden dann folgende Schritte durchgeführt: Zunächst wird ein df erstellt, in dem nur die Daten des aktuellen Profils sind. Dann wird die Drehung des Profils parallel zur X-Achse eingeleitet. Dazu wird die Regressionsgerade durch die Punkte bestimmt um die Steigung zu ermitteln.

```
    i<-1 #Zähvariable aus der Schleife, in diesem Fall 1

    #Alle Daten des iten Profils auslesen und in temporären dataframe schreiben
    coord_proc <- coord[which (coord$pr==prnames[i]), ]
    #nun die Mittelline bestimmen -> lieare Regression
    #Entspricht dem BKS in AutoCAD, allerdings in 2D.
    #Je gerader die Profile, desto genauer. Man könnte auch mit Matritzen arbeiten
,
    #ich denke aber, dass das übertrieben ist.

    #Um die Abweichungen in des ungeraden Profils etwas auszugleichen,
    #wird eine Augleichsgerade zwischen die Punkte geschrieben
    #Dazu wird eine lineare Regression verwendet
    yw <- c(coord_proc$y)
    xw <- c(coord_proc$x)
    fm <- lm(yw ~ xw)

    plot(xw, yw, asp=1)
    abline(fm)
```



Die Steigung der Gerade bestimmt den Winkel um den die Punkte rotiert werden müssen. Dabei müssen zwei Sachen beachtet werden. Ist die Steigung poitiv oder negativ und, von wo wird das Profil betrachet

Es gibt vier Fälle (Sonderfälle sind m = 0)

1. m = - view = N/E -> im Uhrzeigersinn
2. m = - view = S/W -> gegen Uhrzeigersinn
3. m = + view = S/E -> im Uhrzeigersinn
4. m = + view = N/W -> gegen Uhrzeigersinn

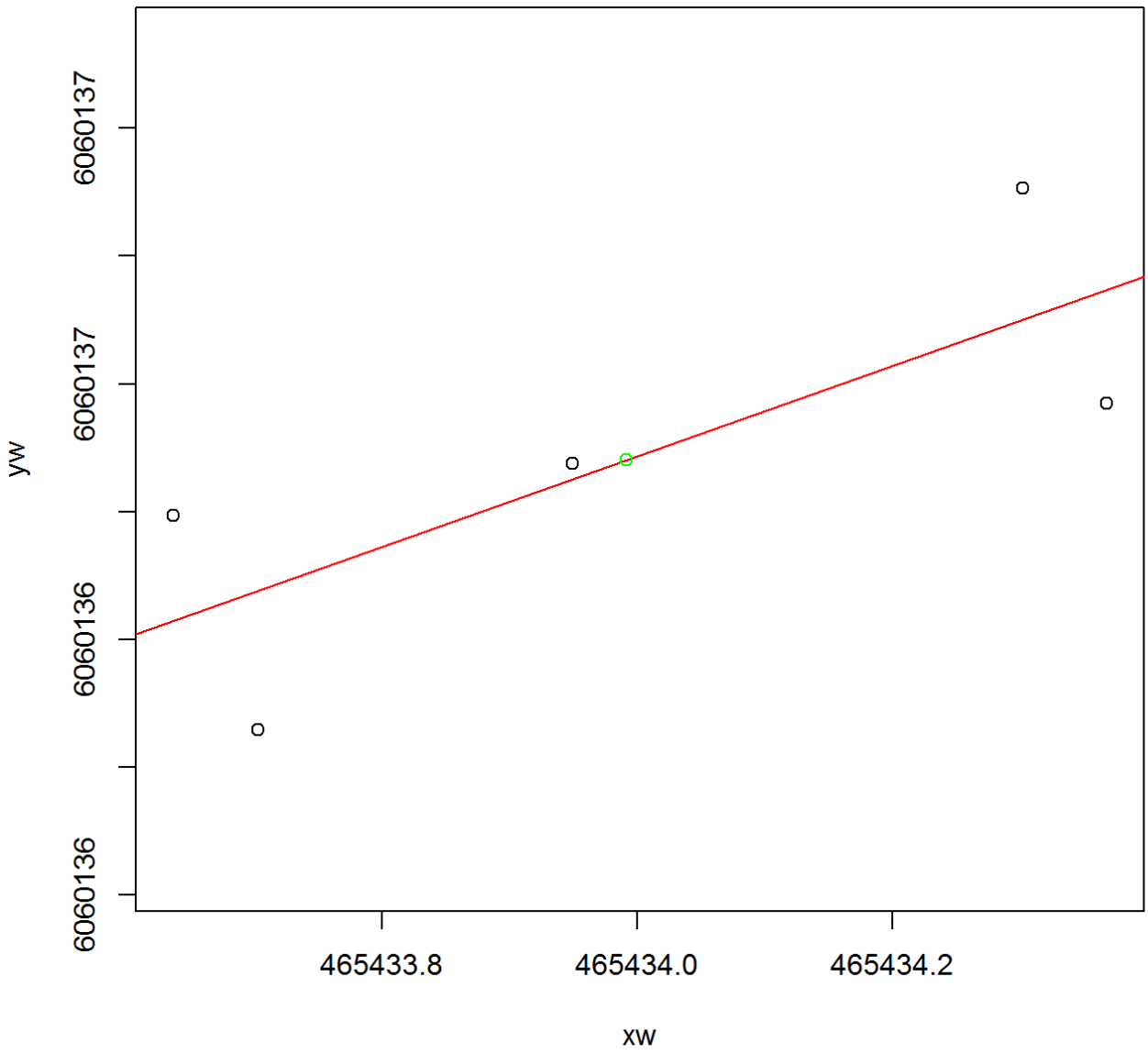
Dafür die Drehwinkel bestimmen (rad -> deg)

```
    #Um die Fotogrammetrienägel korrekt anzeigen zu können, sollen diese gedreht w
erden.
    #Dazu muss der Winkel zwischen der Regressionsgerade und der x-Achse berechnet
werden
    #Steigung der Gerade
    slope <- coef(fm)[2]

        if (slope < 0 && coord_proc$view[1] == "N" ||
        slope < 0 && coord_proc$view[1] == "E"){
      slope_deg <- 180 - abs((atan(slope) * 180) / pi) * -1
    } else if (slope < 0 && coord_proc$view[1] == "S" ||
               slope < 0 && coord_proc$view[1] == "W"){
      slope_deg <- abs((atan(slope)*180)/pi)
    } else if (slope > 0 && coord_proc$view[1] == "S" ||
               slope > 0 && coord_proc$view[1] == "E") {
      slope_deg <- ((atan(slope)*180)/pi)*-1
    } else if (slope > 0 && coord_proc$view[1] == "N" ||
               slope > 0 && coord_proc$view[1] == "W") {
      slope_deg <- 180 - ((atan(slope)*180)/pi)
    } else if (slope == 0 && coord_proc$view[1] == "N" ){
      slope_deg <- 180
    } else if (slope == 0 && coord_proc$view[1] == "N" ){
      slope_deg <- 0
    }

slope_deg
```

```
##       xw
## 160.4551
```

Nun muss der Drehpunkt bestimmt werden. Der liegt mittig aller Punkte *Hier könnte ein Problem entstehen, möglicherweise ist es sinnvoller nur die Maximalwerte zu nehmen, allerdings liegen der Drehpunkt dann nicht auf der Steigungsgeraden, (die könnte man auch nur aus den äußeren berechnen, das muss man dann passend machen)*

```
    #Nun den Drehpunkt bestimmen.
    #X-Wert ist die Mitte zwischen den x-Koordinaten
    center_x <- sum(coord_proc$x)/length(coord_proc$x)
    #Y-Wert
    center_y <- sum(coord_proc$y)/length(coord_proc$y)
    plot(xw, yw, asp=1)
    abline(fm, col="red")
    points(center_x,center_y,col="green")
```



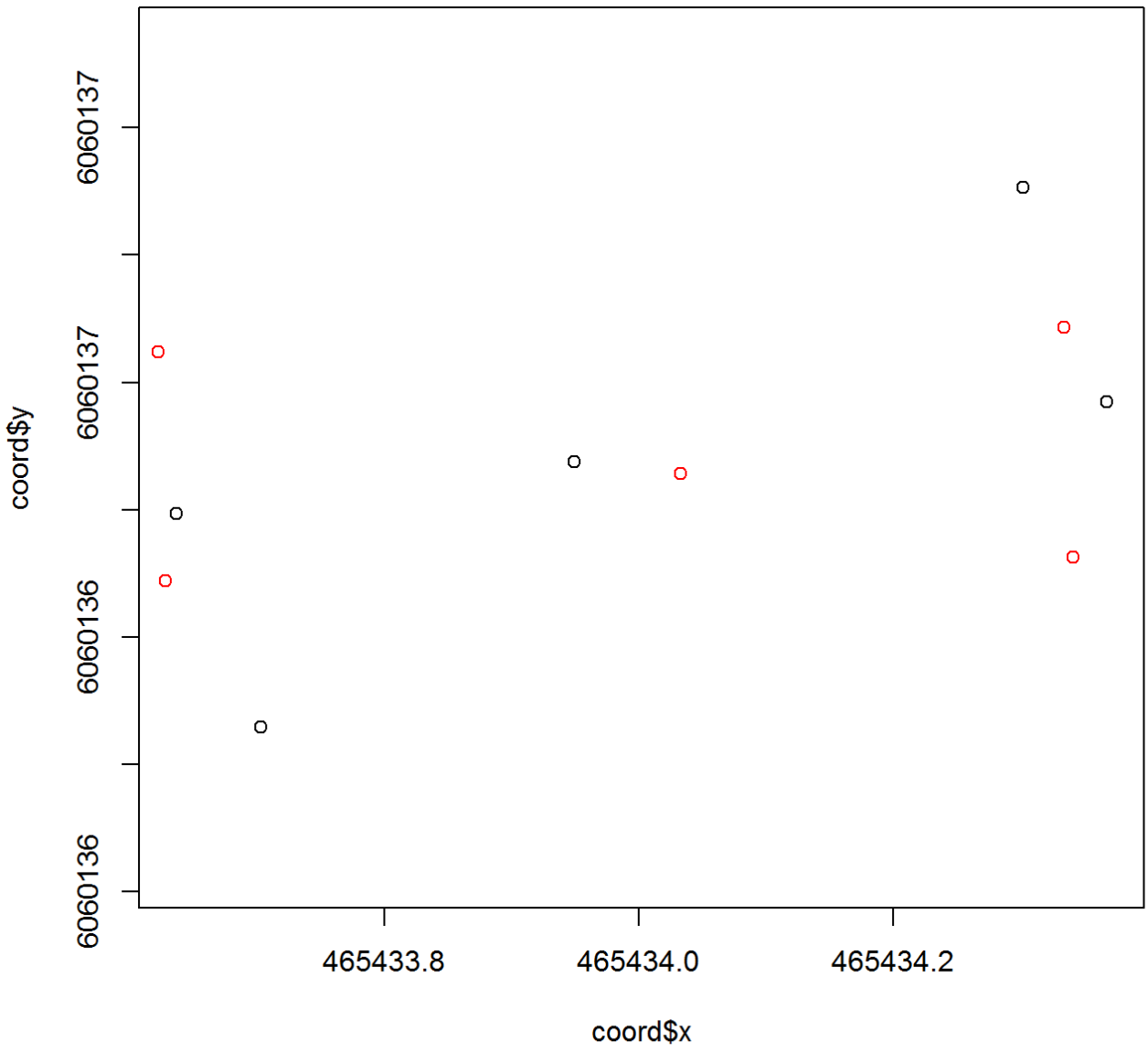
Der nächste Schritt beinhaltet das Drehen (Im Beispiel: Ansicht von Nord). Dafür wird wiederum ein df angelegt, in den die transformierten und rotierten Werte kommen. *http://www.matheboard.de/archive /460078/thread.html (http://www.matheboard.de/archive/460078/thread.html)* Hier besonders wichtig:

coord_procz$[z] + center_y - mean(coord_procz)$

Dadurch liegen die FG-Punkte später mittig über dem Profil

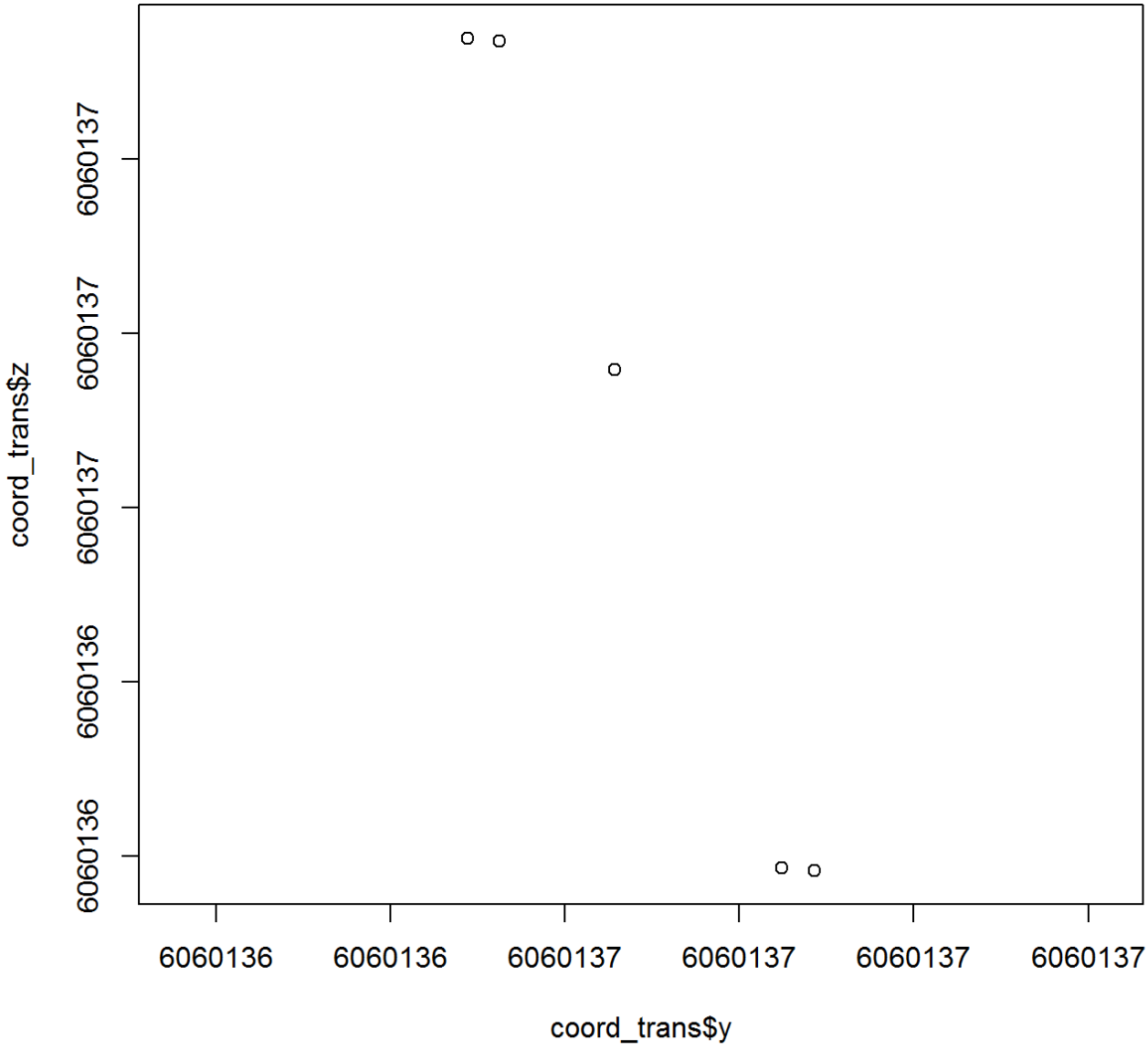
Da es bei langen Profilen einen Fehler gibt, könnte der Nutzer diese in einer Spalte gruppieren, dann muss an dieser Stelle dafür gesorgt werden, dass die Segmente einzeln berechnet und danach nebeneinander dargestellt werden. Das würde die aktuelle Ungenauigkeit einfach lösen

```
coord_trans <- coord_proc
#Die Spalte view fällt raus
coord_trans$view <- NULL
#Für jeden Punkt des Profils mittels translation
#und rotation den neuen Punkt bestimmen
for (z in 1:nrow(coord_proc)){
  coord_trans[z,] <- c(
    center_x + (coord_proc$x[z] - center_x) *
      cos(slope_deg / 180 * pi) - sin(slope_deg / 180 * pi) *
      (coord_proc$y[z] - center_y),
    center_y + (coord_proc$x[z] - center_x) *
      sin(slope_deg / 180 * pi) + (coord_proc$y[z] - center_y) *
      cos(slope_deg / 180 * pi),
    (coord_proc$z[z]+center_y-mean(coord_proc$z)),
    coord_proc$pr[z])
}

plot(coord$x,coord$y,asp=1)
points(coord_trans$x,coord_trans$y, col="red")
```



Jetzt könnten schon die z und y Werte getauscht werden und man hat die Frontalansicht auf das Profil, das ist jedoch nur korrekt, wenn das Profil exakt senkrecht steht. Ob es das tut, lässt bei der Seitenansicht prüfen:

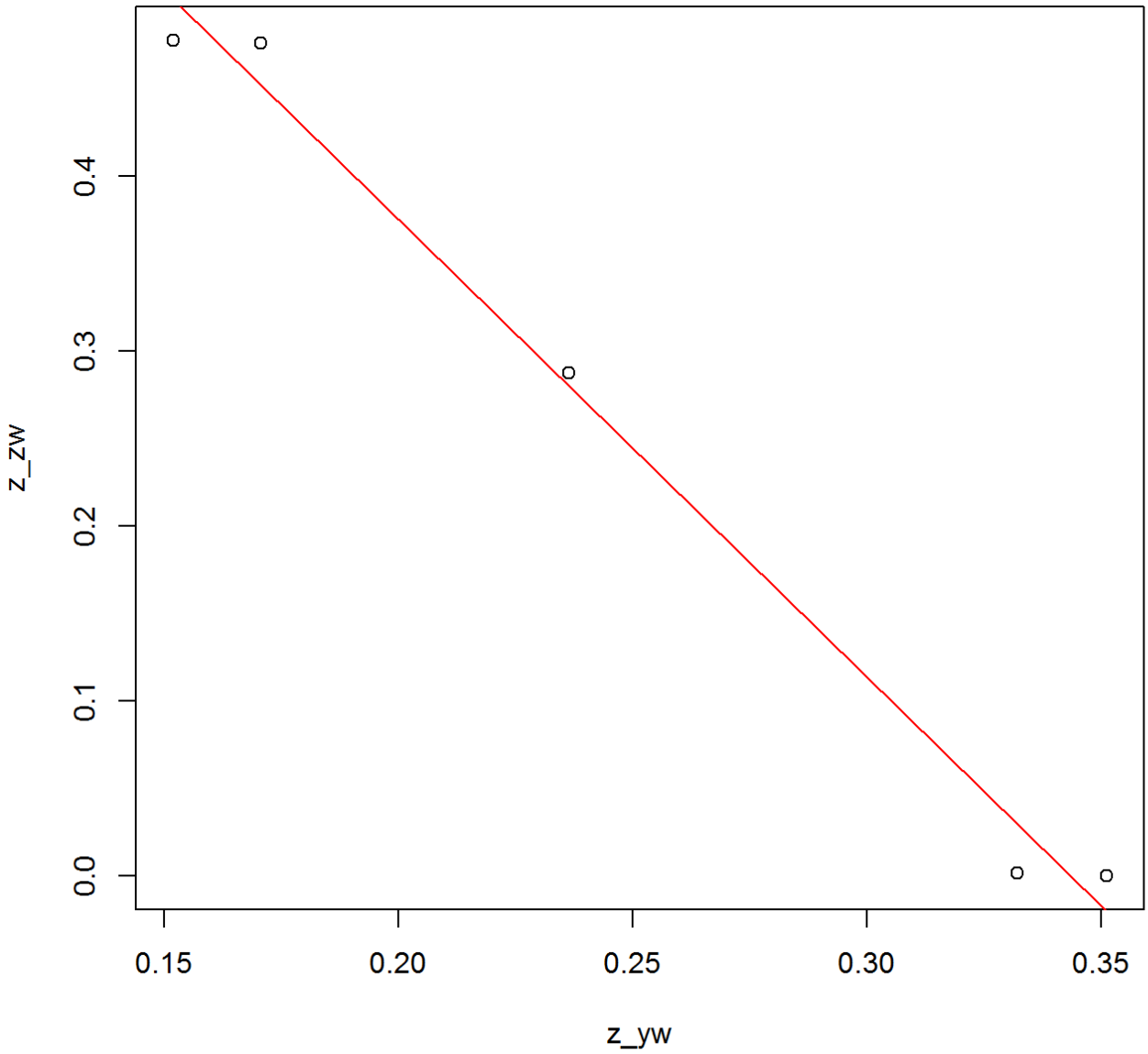
```
plot(coord_trans$y,coord_trans$z,asp=1)
```



Es ist also gekippt. Die Entzerrung geht so nicht, da der Blickwinkel immer orthogonal zu der Z-Achse sein muss, damit keine Verzerrung entsteht . Also müssen die oben durchgeführten Schritte für die Rotation um die z-Achse wiederholt werden.

Hier gab es das Problem, dass nicht mit den echten Koordinatenwerten die lm durchgeführt werden konnte, da gibt es scheinbar ein Problem mit den langen Zahlen. Also wir der Wert an 0 angepasst und die Steigung dann übernommen (*z_yw <-* *c(coord_transy* $- min(c(coord_transy$, *coord_transz)))* $z_z w < -c(coord_transz$ - *min(c(coord_transy*, $coord_transz$*)))*).

```
#Jetzt das ganze für die z-Achse, um eine Kippung des Profils zu minimieren

    z_yw <- c(coord_trans$y -  min(c(coord_trans$y,coord_trans$z)))
    z_zw <- c(coord_trans$z -  min(c(coord_trans$y,coord_trans$z)))

    z_fm <- lm(z_zw ~ z_yw)

    plot(z_yw, z_zw)
    abline(z_fm, col="red")
```

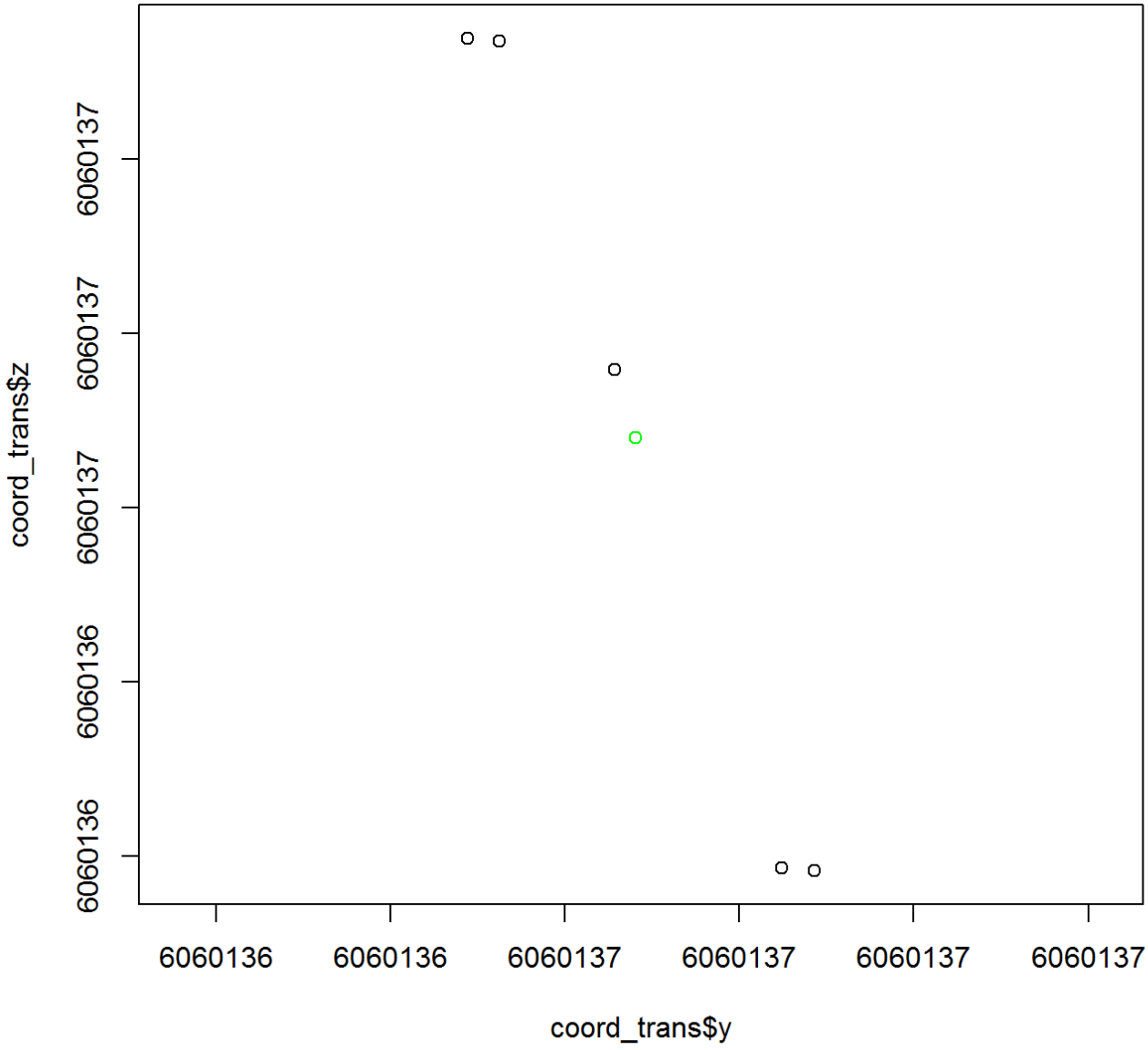


Hier ist die Bestimmung des Roationswinkels einfacher, da immer nur aufgerichtet wird. Also braucht es nur den Fall poisitive oder negative Steigung. Dann die Bestimmung des Drehpunktes

```
    #Steigungswinkel berechnen
    z_slope <- coef(z_fm)[2]

    if (z_slope < 0){

      z_slope_deg <- (90 - abs((atan(z_slope)*180)/pi))*-1
    } else if (z_slope > 0){
      z_slope_deg <- 90 - ((atan(z_slope)*180)/pi)
    } else if (z_slope == 0){
      z_slope_deg <- 0
    }

    z_slope_deg
```

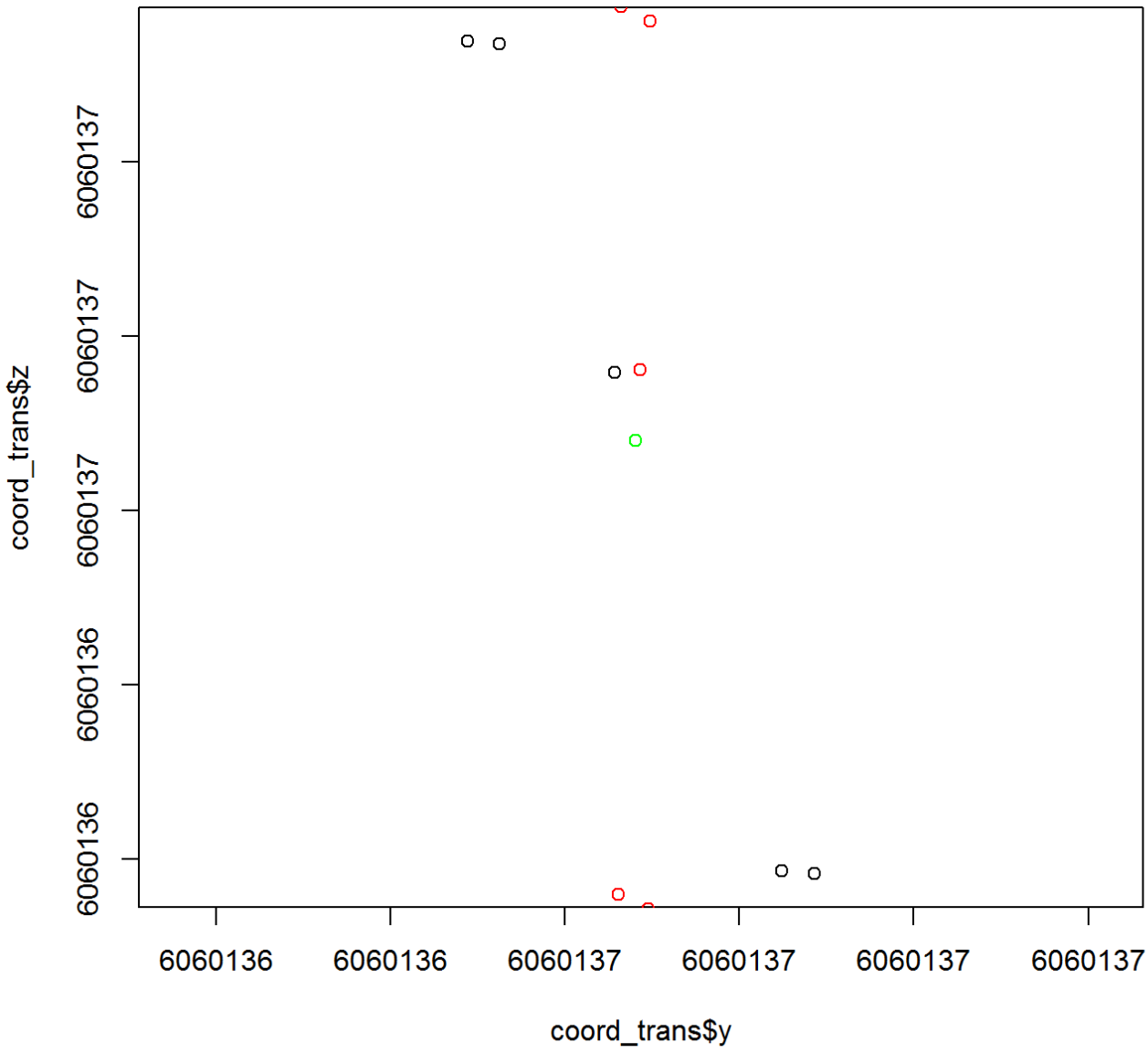
```
##      z_yw
## -20.93203
```

```
    z_center_y <- sum(coord_trans$y)/length(coord_trans$y)
    z_center_z <- sum(coord_trans$z)/length(coord_trans$z)

    plot(coord_trans$y, coord_trans$z, asp=1)
    points(z_center_y,z_center_z,col="green")
```



Und wieder drehen, diesmal um die X-Achse

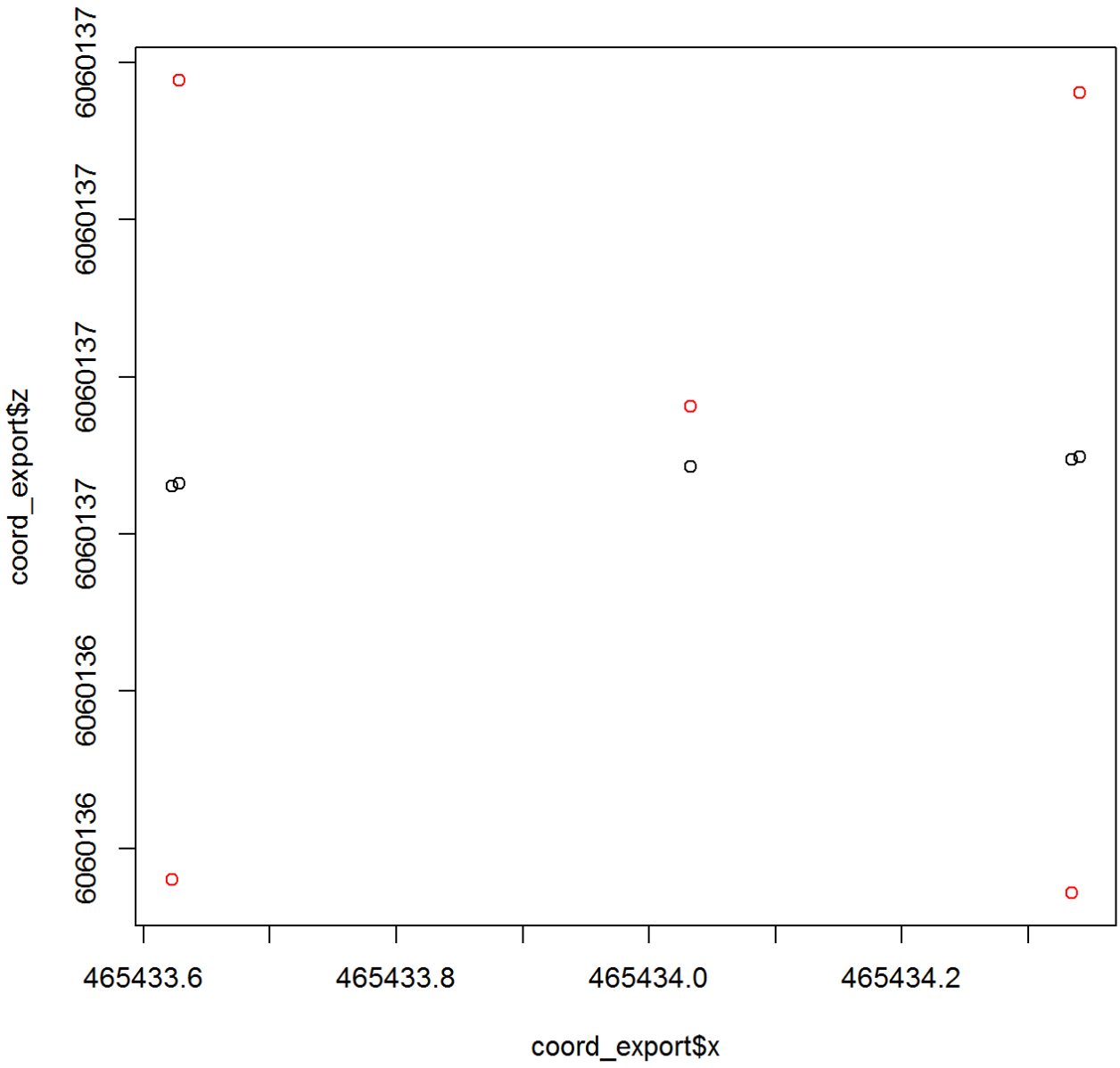
```
    z_coord <- coord_trans

    for (z in 1:nrow(z_coord)){
      z_coord[z,] <- c(
        coord_trans$x[z],
        z_center_y + (coord_trans$y[z] - z_center_y) * cos(z_slope_deg / 180 * pi)
-
          (coord_trans$z[z] - z_center_z) * sin(z_slope_deg / 180 * pi),
        z_center_z + (coord_trans$y[z] - z_center_y) * sin(z_slope_deg / 180 * pi)
+
          (coord_trans$z[z] - z_center_z) * cos(z_slope_deg / 180 * pi),

        coord_trans$pr[z])
      #http://www.matheboard.de/archive/460078/thread.html

    }

    plot(coord_trans$y, coord_trans$z, asp=1)
    points(z_center_y,z_center_z,col="green")
    points(z_coord$y,z_coord$z, col="red")
```




```
    coord_trans <- z_coord
    coord_export[which (coord$pr==prnames[i]), ] <- coord_trans
```

Wenn dies für alle Profile durchgeführt worden ist, folgt die Umwandlung in einen sdf, wobei x und y vertauscht sind, sodass ich die Ansicht auf das Profil bekomme:

```
#Das ganze zu einem Spatialdataframe machen
export <- SpatialPointsDataFrame(coords=coord_export[,c(1,3)],
                                 data = coord_export,
                                 proj4string = (fotogram_pts@proj4string))
plot(coord_export$x,coord_export$z, col="red")
  points(coord_export$x,coord_export$y, asp=1)
```



Weitere Ziele

- Import auch via table (für GIS)
- Prüfen warum bei langen Profilen nicht genau -> Das größte Problem sind die Unterschiede in der Y-Achse. Sind die Werte hier unterschiedlich (nach der Rotation parallel zur X-Achse), funktionier das Klappen nicht richtig. -> Lösung wäre das getrennte umrechnen der Punkte verschiedener Ebenen bzw. das der Einzelsegmente eines langen Profils (am besten mit Gruppierung, dann muss sich der Nutzer um nichts kümmern)
- Welchen Einfluss haben unregelmäßig verteilte Punkte innerhalb des Profiles? -> Sollten nur die Eckpunkte zur Berechnung von Rotationsachse genutzt werden (Reaktion des lm)? -> Nach Tests ist der Einfluss zu vernachlässigen