Map Stitching Using Linear Models in R

Linear Models 2019—Diganta Bhattacharya

1 Solving it Intuitively

1.1 The Problem and its Components

Consider a region small enough so that earth's curvature may be ignored, yet large enough so that Google map cannot fit it in a single screen at high resolution.We cover the region with a number of overlapping screenshots all of the same resolution.Each rectangle is a screenshot (hence is of the size of your monitor). All the screenshots are numbered. To understand the subsequent steps we focus on two overlapping screenshots (say screenshots 1 and 2): The red dots are known locations

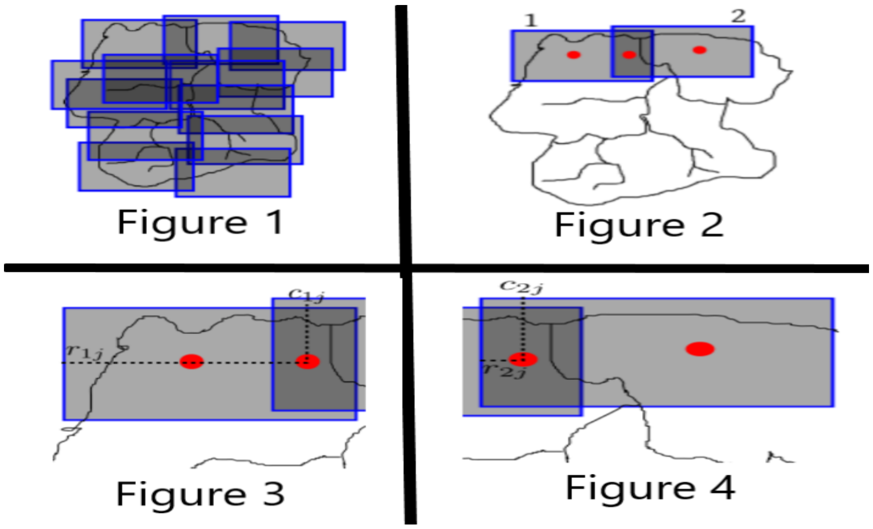


Figure 1: What is happening?

that you can identify at the current resolution of the screenshot. Note that the central location is part of both the screenshots. For each screenshot find the approx pixel coordinate of the location (e.g., by clicking on the centre of each red disk, and reading the mouse coordinates). For example, we measure (r_{1j}, c_{1j}) from screenshot 1,[Here r stands for "row", and c for "column"]. The first subscript is the screenshot number, the second is the location number (we assume that the central red dot in the j-th in the list of the known locations). Similarly, from screenshot 2 we measure (r_{2j}, c_{2j}).Notice that we are using the same j, as it is the same location.

Thus our data set consists of a subset of (r_{ij}, c_{ij})'s for $i = 1, \cdots$, number of screenshots, and $j = 1, \cdots$, number of locations. Of course, not all (i, j) pairs occur in the data, since the $j - th$ location may not show up in the $i - th$ screenshot. Let (μ_i, λ_i) be the true position of the $i - th$ location (w.r.t. some global coordinate system). We can set up a linear model to estimate (μ_i, λ_i)'s from the data.

Note 1. We have three cases in total-

- **NO ROTATION SAME RESOLUTION**
- **NO ROTATION DIFFERENT RESOLUTION**
- **ROTATION AND DIFFERENT RESOLUTION**

1.2 The Basic Solution

Let us solve the very basic case first. When everything is of the same resolution and there are no rotations. Then this means essentially we just have to do some basic x and y shifting. The important things in this setup are :

- The Local Co-ordinate of each location in each picture.
- The Global Co-ordinate of each of the locations.
- The Picture/Screenshot Number for which we will find the local locations.

And in order to make our lives easier in the data entry part we will use the following linear model:- Let us assume there is a global coordinate system with the same scale that is in the screenshots. Now, let the global coordinates of location j be (x_j, y_j). Also, let the origin of the ith screenshot be at (a_i, b_i) with respect to the global system. Then, ideally, if r_{ij}, c_{ij} is as described in the problem , then-

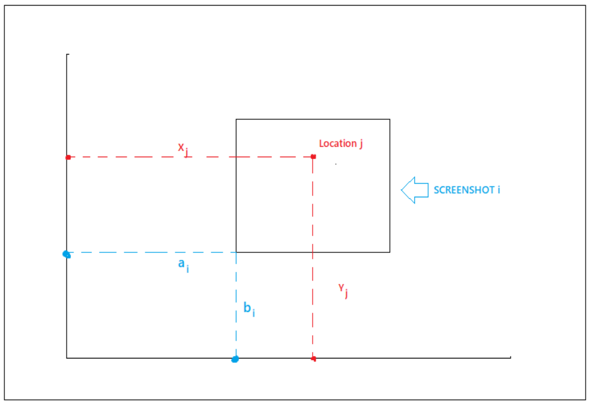


Figure 2: Nomenclature of the points

$$r_{ij} = x_j - a_i \tag{1}$$

$$c_{ij} = y_j - b_i \tag{2}$$

But as there could be error in marking the locations in the screenshots, we don't get the exact value for r_{ij}, c_{ij}. Hence, we assume there are errors which follow normal distribution with mean zero independently for both the directions X and Y.

According to the discussion above, we have for the most basic model-

$$r_{ijk} = x_j - a_i + \varepsilon_{ijk} \tag{3}$$

where i corresponds to the screenshot number, j corresponds to the location number, k is for the number of times the same location is selected in the same screenshot. we assume that the errors follow $N(0, \sigma^2)$ and are independent.
Similarly, for the $Y-$direction,

$$c_{ijk} = y_j - b_i + \varepsilon_{ijk} \tag{4}$$

Where i, j, k have the same interpretations and errors follow iid $N(0, \sigma^2)$.

Note 2. Note: We will also assume that the errors in the X direction are independent with the errors in the Y direction.

1.3 The Approach for Homothety and Rotations

First, we start with two pictures and then we will try to extend it to more than 2 pictures. For each of the things we would try to find a property in the pictures which should remain the same if they are pictures of the same thing, even if the do not have the same resolution or they might even be rotated. So we will be looking for those properties of a picture which do not change under rotation or magnification. Let us look at two same pictures which have 3 intersection points. Firstly, notice

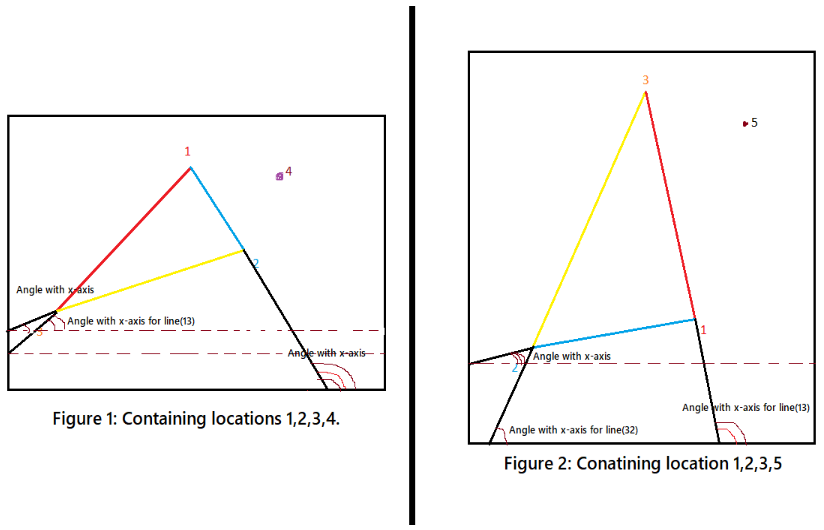

1
4
2
3
Angle with x-axis
Angle with x-axis for line(13)
Angle with x-axis
Figure 1: Containing locations 1,2,3,4.
3
5
1
2
Angle with x-axis
Angle with x-axis for line(13)
Angle with x-axis for line(32)
Figure 2: Conatining location 1,2,3,5

Figure 3: Two pictures with no. of int=3

that if the pictures are not in the same scale then lateral shifting and rotation makes no sense since

the angles might get enlarged due to magnification.
Question- Say *figure 1* is the Master picture then *figure 2* is connected with *figure 1* as seen. How can we compare the size of these two pictures given that I know 3 intersecting locations and their positions in both the pictures? What is the property which remains invariant if one picture is truly a magnified version of another?
Answer- If we know 3 intersecting locations,it also means the 3 lines joining them are also known, we can also calculate their distances.The ratio of the length of the lines in the master picture and the second picture should be invariant if the magnification was uniform. So we found our first property which will help us deal with homothety.
What Do I Mean?
Say $l_{1,12}$ denote the length of the line 12 in picture1, similarly $l_{2,12}$ be the length of the line 12 in picture 2. Then the ratio $= \frac{l_{2,12}}{l_{1,12}}$ should be invariant for all such common lines in the ideal case. The ratios,

- $\frac{l_{2,12}}{l_{1,12}}$
- $\frac{l_{2,23}}{l_{1,23}}$
- $\frac{l_{2,31}}{l_{1,31}}$

should be equal and they should give us the magnification factor of every picture wrt the master picture which is the first picture in this case. Now we will apply our linear model knowledge to estimate these ratios for each picture.
Linear Model and its Parameters

The linear model should involve all the intersecting lines, and through each model we get the magnification factor for each image.

$$l_{1,j,lk} = \beta_{1,j} + \varepsilon_{1,j,lk} \tag{5}$$

where j corresponds to the screenshot number whose magnification with respect to master picture is to be determined. $\beta_{1,j}$ is this required magnification ratio. , lk corresponds to the line which is examined in both the screenshots. We assume that the errors follow $N(0, \sigma^2)$ and are independent. So if there are n intersecting points there would be $\binom{n}{2}$ lines which would be common n both the master picture and this picture.

Now, after homothety factors for each of the picture is calculated we reduce each image to the magnification level of the master image in this case *figure 1*.Now, the angles between the same lines should be same. For rotation we need to find another such property which remains invariant.

Question- If each picture has to be rotated by some degrees so that everything becomes parallel, how do we find the value of that degree? We only know the common intersection points with the master image.
Answer The angle that the common lines make with the x axis in the master picture and in the second picture share a relation. For each line the difference in the angle it subtended with the

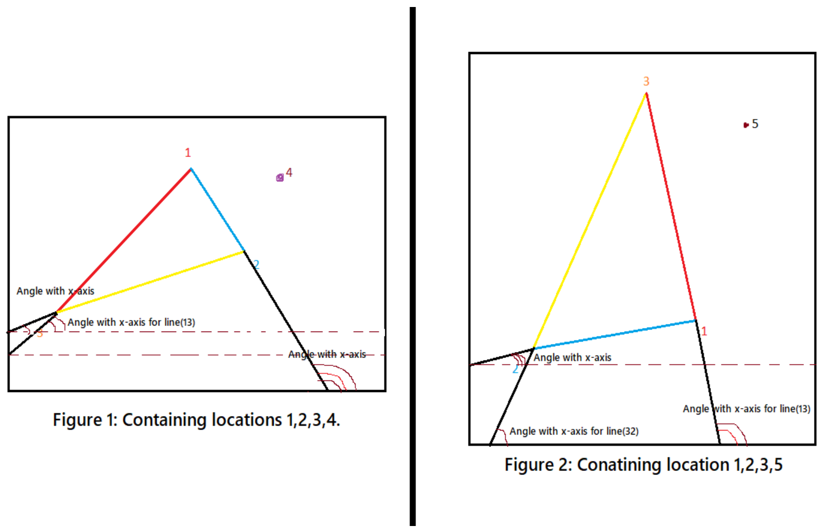

1
4
2
Angle with x-axis
Angle with x-axis for line(13)
3
Angle with x-axis
Figure 1: Containing locations 1,2,3,4.
3
5
1
Angle with x-axis
2
Angle with x-axis for line(13)
Angle with x-axis for line(32)
Figure 2: Conatining location 1,2,3,5

Figure 4: Two pictures with no. of int=3

x-axis of the master image and the second image should be invariant and it should give you the required angle for rotating the second picture in order to make everything parallel.

What Do I Mean?

Say $m_{1,12}$ denote the slope of the line 12 in picture1 with the x-axis[making an angle $tan^{-1}(m_{1,12})$], similarly $m_{2,12}$ be the slope of the line 12 in picture 2 with the x-axis[making an angle $tan^{-1}(m_{2,12})$]. Then the difference between the two calculated angles would be $= [tan^{-1}(m_{1,12}) - tan^{-1}(m_{2,12})]$ should be invariant for all such common lines in the ideal case. The differences,

- $[tan^{-1}(m_{1,12}) - tan^{-1}(m_{2,12})]$
- $[tan^{-1}(m_{1,23}) - tan^{-1}(m_{2,23})]$
- $[tan^{-1}(m_{1,31}) - tan^{-1}(m_{2,31})]$

should be equal and they should give us the angle with which every picture should be rotated so that every becomes parallel w.r.t the master picture which is the first picture in this case. Now we will apply our linear model knowledge to estimate these differences for each picture.

Linear Model and its Parameters

The linear model should involve all the intersecting lines, and through each model we get the rotation factor for each image. Let us denote by $\Theta_{i,jkl}$ the angle difference between the slope of line jk in screenshot i and the master picture. Then,

$$\Theta_{i,jk} = \theta_{1,i} + \varepsilon_{i,jk} \tag{6}$$

where i corresponds to the screenshot number whose rotation angle with respect to master picture is to be determined. $\theta_{1,i}$ is this required rotation angle for screenshot i. , jk corresponds to the line which is examined in both the screenshots. We assume that the errors follow $N(0, \sigma^2)$ and are independent. So if there are n intersecting points there would be $\binom{n}{2}$ lines which would be common in both the master picture and this picture.

2 How to use the code

There are some things one should follow in order to get the correct response from the code.They are listed below.

- During data entry the **name** of the **Locations** as well as the screenshot number should be given as integers.
- Same locations should be marked with the same integer in different screenshots.
- Initially the code will ask you to give how many location points do you have in total.You should name the locations according to the no. of location points you have, i.e *The integer corresponding to the last location should be same as the number of locations you have in total.*.We are using consequtive numbers for marking locations.
- All the screenshots should be saved in the working directory an the code will automatically show them to you one by one as you fill up how many locations you have in each screenshot.
- **You should not mark a point more than once in a screenshot.If you want to estimate an area like say BHOS, then mark the vertices of BHOS in the correct orientation in all the screenshots.** This means rather than estimating a rectangle by its centre's aprroximation I would estimate its vertices, essentially giving me the whole shape.
- If some screenshot has only one or two locations then try to avoid giving them since the estimates might be bad for those and they may make the whole model bad.
- While clicking each location you have to name that location and then mark the next one till you get to the next screenshot.
- Write $\boxed{X = linalg()}$ and then the code will run giving you the results. And X will store all the computations that can be checked.**To check the values, use X[1] to x[6] to view the computations.**

3 Other Details in the R code

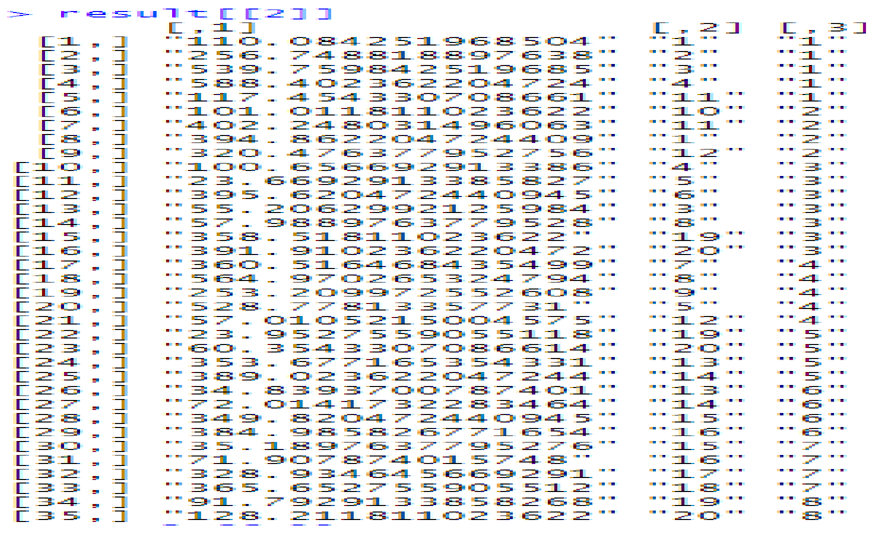
```
linalg=function()
  {
   ###############################################
   #####Installing required Packages############
   ###############################################
   print("Let's start by installing the required libraries.")
   if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
   BiocManager::install("EBImage")
   ##########################################################
   ##########My own locator which takes names###############
   ##########################################################
   plot.locator<-function(item.to.plot, num.points)
     {
```

```
   plot(item.to.plot)
   xs.list<-c(NULL)
   names.list<-c(NULL)
   ys.list<-c(NULL)
   for( I in 1:num.points)
     {
      message("Name this point,select next point")
      #Enter Warning Message Here!
      response <- readline(prompt="Enter name: ")
      location<-locator(1,type="p", par(pch=16,col="blue"))
      xs.list<-c(xs.list,location$'x')
      ys.list<-c(ys.list,location$'y')
      names.list<-c(names.list,response)
     }
   locations<-list(x=xs.list, y =ys.list, name =names.list)
   return(locations)
  }
##########################################################
############## Intersection Function ##################
##########################################################
set.intersection <- function(a, b)
  {
   intersect <- vector()
   for (i in 1:length(a))
     {
      if (a[i] %in% b)
        {
         intersect <- append(intersect, a[i])
        }
     }
   return(intersect)
  }
##########################################################
##############Data Input in R with names################
##########################################################
input<-function()
  {
   n = readline(prompt="How many pictures you want to merge?")
   num.pictures = as.integer(n)
   if(num.pictures < 1)
     {
      print("That was not expected.")
     }
   print("Save each picture in your working directory. Your working directory is--")
   print(getwd())
   print("Later you are supposed to enter the names of the images as saved, e.g,
       image.png ")
   k = readline(prompt="How many location points you have in total?")
   num.locations = as.integer(k)
   print("Make sure to name the locations as consecutive integers starting from 1.
       One particular location must be named the same every time.")

   X_matrix = matrix( ,nrow = num.pictures,ncol = num.locations)
   Y_matrix = matrix( ,nrow = num.pictures,ncol = num.locations)
   image<-matrix(,nrow= num.pictures,ncol = 1)
   information1<-NULL
   information2<-NULL
   location.list=list()
   for(i in 1:num.pictures)
     {
      image[i]<-readline(prompt="Enter the name of the image as saved in working
          directory including the .png or .jpeg part.")
      #Enter Warning Message Here!
      image<-readImage(image[i])
      plot(image)
      num.points=readline(prompt ="Enter the number of points you want to
          mark.Marking 0 is not going to help.")
      #Enter Warning Message Here!
      num.points=as.integer(num.points)
      if(num.points <1)
        {
         print("That's not an expected response. I am afraid you have to run the
             function again. Press esc *~*")
        }
      info<-plot.locator(image,num.points)
      location.list[[i]]=info$name
      for(j in 1:(length(info$name)))
        {
         information1<-rbind(information1,c(info$x[j],info$name[j],i))
         information2<-rbind(information2,c(info$y[j],info$name[j],i))
         location<-as.integer(info$name[j])
         X_matrix[i,location]=info$x[j]
         Y_matrix[i,location]=info$y[j]
        }
     }
   Information_List = list(information1,information2,X_matrix,Y_matrix,location.list)
   return(Information_List)
  }
############################################################
############## Basic Homothety Shifts ###################
############################################################
Homothety.func<-function(A)
  {
   X_matrix=A[[3]]
   Y_matrix=A[[4]]
   n=nrow(X_matrix)
   beta=matrix(rep(1,times=n*n),nrow=n)
   Basis_set=NULL
   Residual_set=NULL
   for(i in 2:n)
     {
      a=as.integer(A[[5]][[1]])
      b=as.integer(A[[5]][[i]])
      if(length(set.intersection(a,b))<2)
        {
         Residual_set=c(Residual_set,i)
        }
      else
      {
       Basis_set=c(Basis_set,i)
       h1=NULL
       h2=NULL
       S=set.intersection(a,b)
       count=0
       for(l in S)
         {
          for(k in S)
            {
             if(l<k)
               {
                h1<-c(h1,(((X_matrix[1,l])-(X_matrix[1,k]))^2 +
                    (Y_matrix[1,l]-Y_matrix[1,k])^2)^(1/2))
                h2=c(h2,(((X_matrix[i,l])-(X_matrix[i,k]))^2 +
                    ((Y_matrix[i,l])-(Y_matrix[i,k]))^2)^(1/2))
               }
            }
         }
       beta[1,i]=mean(h2/h1)
      }
   }
 for(i in Basis_set)
   {
    for(j in Residual_set)
      {
       a=A[[5]][[i]]
       b=A[[5]][[j]]
       if(length(set.intersection(a,b))<2)
         {
          beta[i,j]=1
          #Enter Warning Message Here!
         }
      else
         {
          Basis_set=c(Basis_set,j)
          Residual_set=Residual_set[which(Residual_set !=j)]
          S = set.intersection(a,b)
          S=as.integer(S)
          h1<-NULL
          h2<-NULL
          h=NULL
          count=0
          for(l in S)
            {
             for(k in S)
               {
                if(l<k)
                  {
                   h1<-c(h1,(((X_matrix[i,l])-(X_matrix[i,k]))^2 +
                       (Y_matrix[i,l]-Y_matrix[i,k])^2)^(1/2))
                   h2=c(h2,(((X_matrix[j,l])-(X_matrix[j,k]))^2 +
                       ((Y_matrix[j,l])-(Y_matrix[j,k]))^2)^(1/2))
                  }
                }
             }
            beta[i,j]=mean(h2/h1)
            beta[1,j]=beta[1,i]*beta[i,j]
           }

        }
     }
    connected_pic=factor(Basis_set)
    connected=levels(connected_pic)
    result=list()
    result[[1]]=beta
    result[[2]]=connected
    return(result)
   }
  ##########################################################
  ##################### Rotations #########################
  ##########################################################

Rotations.func<-function(A)
   {
    X_matrix=A[[3]]
    Y_matrix=A[[4]]
    n=nrow(X_matrix)
    beta2=matrix(rep(1,times=n*n),nrow=n)
    Basis_set2=NULL
    Residual_set2=NULL
    for(i in 2:n)
      {
       a=as.integer(A[[5]][[1]])
       b=as.integer(A[[5]][[i]])
       if(length(set.intersection(a,b))<2)
         {
          Residual_set2=c(Residual_set2,i)
         }
       else
         {
          Basis_set2=c(Basis_set2,i)
          h1=NULL
          h2=NULL
          S=set.intersection(a,b)
          count=0
          for(l in S)
            {
             for(k in S)
               {
                if(l<k)
                  {
                   h1<-c(h1,(180/pi)*(atan(((Y_matrix[1,l])-(Y_matrix[1,k]))/(((X_matrix[1,l])-(X_matri
                   h2<-c(h2,(180/pi)*(atan(((Y_matrix[i,l])-(Y_matrix[i,k]))/(((X_matrix[i,l])-(X_matri
                  }
               }
            }
          beta2[1,i]=mean(h2-h1)
        }
      }
    for(i in Basis_set2)
      {
       for(j in Residual_set2)
        {
         a=A[[5]][[i]]
         b=A[[5]][[j]]
         if(length(set.intersection(a,b))<2)
           {
            beta2[i,j]=1
            #Enter Warning Message Here!
           }
        else
           {
            Basis_set2=c(Basis_set2,j)
            Residual_set2=Residual_set2[which(Residual_set2 !=j)]
            S = set.intersection(a,b)
            S=as.integer(S)
            h1<-NULL
            h2<-NULL
            h=NULL
            count=0
            for(l in S)
              {
               for(k in S)
                 {
                  if(l<k)
                    {
                     h1<-c(h1,(180/pi)*(atan(((Y_matrix[1,l])-(Y_matrix[1,k]))/(((X_matrix[1,l])-(X_ma
                      h2<-c(h2,(180/pi)*(atan(((Y_matrix[i,l])-(Y_matrix[i,k]))/(((X_matrix[i,l])-(X_m
                    }
                 }
              }
            beta2[i,j]=mean(h2/h1)
            beta2[1,j]=beta2[1,i]*beta2[i,j]
           }

        }
      }
    connected_pic2=factor(Basis_set2)
    connected2=levels(connected_pic2)
    result=list()
    result2[[1]]=beta2
    result2[[2]]=connected2
    return(result2)
  }
#########################################################
############## Basic Parallel Shifts ####################
#########################################################
A<-input()
result<-Homothety.func(A)
beta=result[[1]]
class(beta)="numeric"
```

```
 X=A[[1]]
 Y=A[[2]]
 class(X)<-"numeric"
 class(Y)<-"numeric"
 n=nrow(X)
 ###########################################################
 ################## Adjusting Homothety ###################
 ###########################################################
 for(i in 1:n)
   {
    no.picture=X[i,3]
    X[i,1]=X[i,1]/beta[1,no.picture]
    Y[i,1]=Y[i,1]/beta[1,no.picture]
   }
###########################################################
################### Adjusting Rotations ##################
###########################################################
 for(i in 1:n)
   {
    no.picture=X[i,3]
    X[i,1]= X[i,1]cos(beta2[1,no.picture]) + Y[i,1]sin(beta2[1,no.picture])
    Y[i,1]= -X[i,1]sin(beta2[1,no.picture]) + Y[i,1]cos(beta2[1,no.picture])
   }
 ###########################################################
 ###########################################################
 ###########################################################

 location.f<-factor(X[,2])
 picture.f<-factor(X[,3])
 Xshift<-lm(X[,1]~location.f + picture.f -1)
 Yshift<-lm(Y[,1]~location.f + picture.f -1)
 Locations<-levels(location.f)
 l=paste("location.f",Locations,sep="")
 Pictures=levels(picture.f)
 N=length(Locations)
 u=coef(Xshift)[l]
 class(u)<-"numeric"
 v=coef(Yshift)[l]
 class(v)<-"numeric"
 v=rep(max(v)+5,times=N)-v
 plot(u,v,asp=1)
 text(u,v-5,Locations)
 Xerror=coef(summary(Xshift))[1:N,2]
 Yerror=coef(summary(Yshift))[1:N,2]
 location.matrix=cbind(Locations,u,v,Xerror,Yerror)
 if((max(Xerror)>10) | (max(Yerror)>10))
   {
    print("The fit is not good judging from the errors. There might have been mistakes
        in marking the points.")
   }
 print("If there are some disconnected pictures they will be printed below- ")
 print(result[[2]])
 class(location.matrix)<-"numeric"
 colnames(location.matrix)<-c("Locations","X-coordinates","Y-coordinates","Error.Xdirection","Error.Y
 return(as.matrix(location.matrix))
}
```

3.1 Data Entry by the User

For this we will have to use the *Locator* package in R. For my version I have tweaked the original *Locator* so that the data entry process is easier.I will not be using the locator function but my own **plotlocator**. What my **plotlocator** does is that whenever the user clicks on the desired points it will store the x and y coordinates of the click and also store the name of the point as given by the user. All this information will be stored in 3 different matrices which will be called later on to fit the linear models.
So basically after data entry the Xmatrix,Ymatrix and the Namematrix would look somewhat like this-



```
> result[[2]]
      [,1]                  [,2]  [,3]
 [1,] "110.084251968504"    "1"   "1"
 [2,] "256.748818897638"    "2"   "1"
 [3,] "539.759842519685"    "3"   "1"
 [4,] "588.402362204724"    "4"   "1"
 [5,] "117.454330708661"    "11"  "1"
 [6,] "101.011811023622"    "10"  "2"
 [7,] "402.248031496063"    "11"  "2"
 [8,] "394.862204724409"    "1"   "2"
 [9,] "320.476377952756"    "12"  "2"
[10,] "100.656692913386"    "4"   "3"
[11,] "23.669291338582 7"   "5"   "3"
[12,] "395.620472440945"    "6"   "3"
[13,] "55.2062992125984"    "3"   "3"
[14,] "57.9889763779528"    "8"   "3"
[15,] "358.51811023622"     "19"  "3"
[16,] "391.910236220472"    "20"  "3"
[17,] "360.516468435499"    "7"   "4"
[18,] "564.970265324794"    "8"   "4"
[19,] "253.209972552608"    "9"   "4"
[20,] "528.77813357731"     "5"   "4"
[21,] "57.0105215004575"    "12"  "4"
[22,] "23.9527559055118"    "19"  "5"
[23,] "60.3543307086614"    "20"  "5"
[24,] "353.677165354331"    "13"  "5"
[25,] "389.023622047244"    "14"  "5"
[26,] "34.8393700787401"    "13"  "6"
[27,] "72.0141732283464"    "14"  "6"
[28,] "349.820472440945"    "15"  "6"
[29,] "384.985826771654"    "16"  "6"
[30,] "35.1897637795276"    "15"  "7"
[31,] "71.90787401574 8"    "16"  "7"
[32,] "328.934645669291"    "17"  "7"
[33,] "365.652755905512"    "18"  "7"
[34,] "91.7929133858268"    "19"  "8"
[35,] "128.211811023622"    "20"  "8"
```

Figure 5: Data matrix

This is the XMatrix for the case with no rotations and same resolution which gives columnwise the coordinates, screenshot number and the location name.

3.2 Handling the Basis And Residual Sets

First we need to fix a picture as the Master Picture, with respect to which we will change the size of all the other images. For that we need to know which all pictures are connected with the Master picture(connected means the pictures share some intersection points$\geq$ 2) and make them form a set called **Basis Set**.
From the elements of the **Basis Set** we will traverse the rest of the pictures, meaning for each element of the **Basis Set** we will find all those pictures which are connected with it. We call this **Residual Set**. If there are picture which are neither in the **Basis Set** or the **Residual Set** then they do not provide any information and the code should eliminate those picture from the data,

which is precisely what this code will do. It will take the first picture as the Master picture and then find all the pictures which are attached to this picture and call them the **Basis Set**. Now for all the elements of the basis set we will find out which pictures are connected to them and not the Master picture directly and we call them the **Residual Set**. We then print everything that is in the **Basis Set** and **Residual Set** as they will provide us information about the original location of the locators.

3.3 Applying Linear Models Directly on the Homothety/Rotation Part vs BFS then taking means.

Now, if you observe this algorithm carefully you would see that in the case of homothety we can apply linear models directly without going through all the fiasco of **Basis Set** and **Residual Set**. In that case we would have a linear model like this for example:

$$D_{i,jk} = \gamma_{1,i} \times d_{1,jk} + \varepsilon_{i,jk} \tag{7}$$

See, if we take log on both sides this becomes a linear model. And hence we would be able to run lm on it directly. $D_{i,jk}$ is the length of the jk line in i-th picture and $d_{1,jk}$ is the length of the jk line in the master picture.$\gamma_{1,i}$ is the magnification factor of the i-th image.

But, in case of rotation, it is not easy to fit a linear model for the whole thing. So for this reason I will avoid using linear models directly. I would ask the user to give the picture with the most number of connections as the master image. And then I would **compare** the magnification level and rotation angle of all the images with this master image. If there are direct connections I will not consider the angles I get from between those direct connections because then I would also have to include the condition $\theta_1 + \theta_2 + \theta_3 = \pi$.
Similarly, in case of homothety if I directly apply the linear model mentioned above I will be using the information I get from the interactions of two screenshots in the basis set, which may lead to a bad result since I am not using the information that my Master Image is most accurately marked. Also, there would be some lines about which my code won't have any information if I directly run ANNOVA on it, I would still have to use BFS.

3.4 How do I understand if someone has accidentally marked some points wrongly?

There can be two possible ways for an error to happen.

- Someone has intentionally altered the orientations of the locations in more than one of the screenshots. In that case there is no possible way for my code to detect that there was an accidental error.
- If someone has marked some locations inaccurately or say some locations(few) have their orientations altered in some(few) screenshots. Then for the basic model my code will check

the maximum of the residuals and report it. And for the other models it would compare the whole thing with corresponding LAD[**Least Absolute Deviation**] estimates which would calculate everything about **Medians** , thus cancelling all out-lier effects. If the difference between these two estimates(final global coordinates) are high, my code will report it.

So what should happen is that if someone accidentally has marked some points or locations in the wrong orientation, the mean would minimize the squared error,but the mean would be affected by these outliers. Now if we try to fit the LAD regression then the median will minimize the mean deviation, and median will not be affected by a small number of outliers. And both of these estimates should be quite close if there was no error in data input by the user. So any absurd deviation in these estimates should give the user a warning that there might be sme issues with data entry.

4 Conclusion

In the given data set since there were only two intersecting points in most of the screenshots the LAD and the LS estimates were nearly same. So there should be no error if the orientations are correctly marked.
If there was any error in data entry this code will specify for which part the estimates are not good , i.e if the error is high in homothety or during rotations or during parallel shiftings.The code reports the deviations of the LS and LAD estimates, and a indicator will go off if their estimates have a difference of more than 10 pixels.

4.1 Packages to be Installed

The packages that should be installed in R are :

- OpenImageR
- magick
- imager
- EBImage(BioConductor Package)

5 Results on the Given data Set

- CASE 1:NO ROTATIONS SAME RESOLUTION.
  The data set can be found here- https://www.isical.ac.in/ arnabc/linmod/projects.html

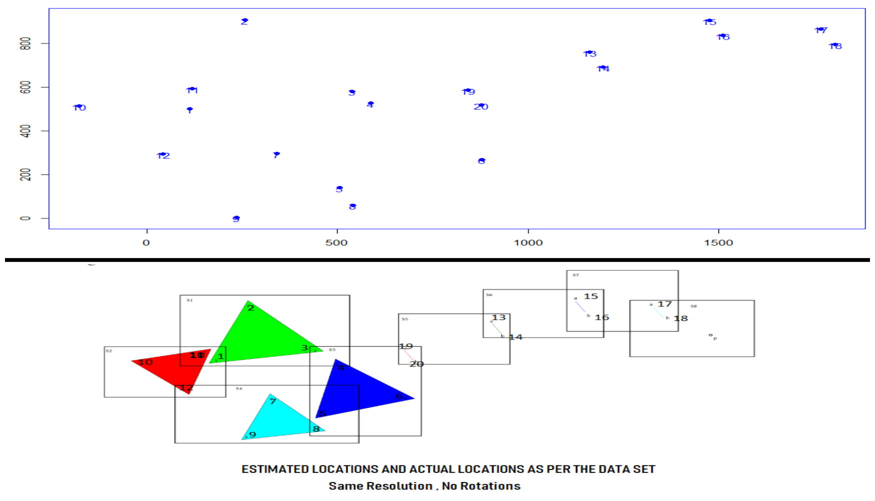


Figure 6: Final Locations

- CASE 2:NO ROTATIONS DIFFERENT RESOLUTION.
  The data set can be found here- https://www.isical.ac.in/ arnabc/linmod/projects.html

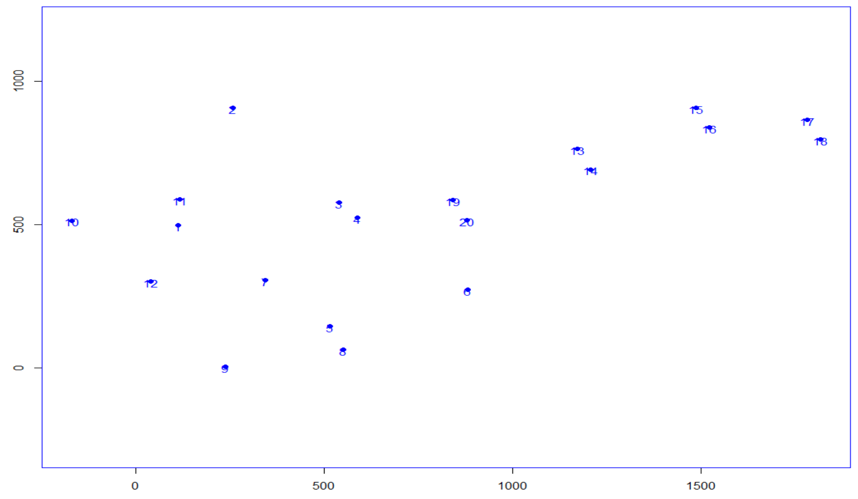


Figure 7: Final Locations

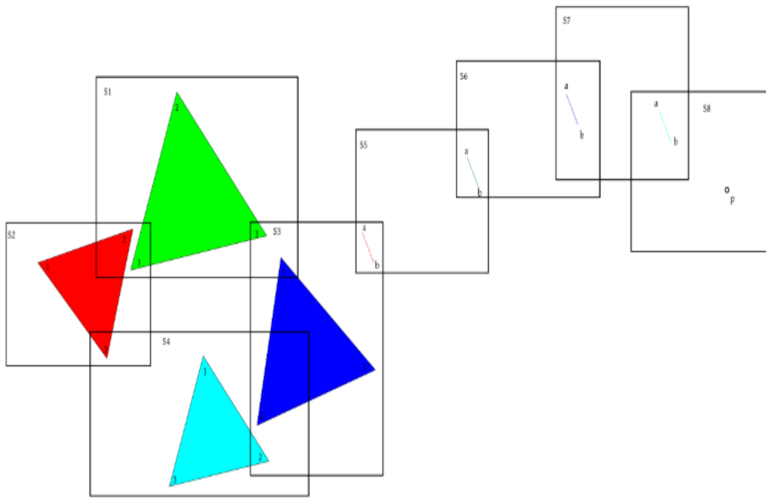
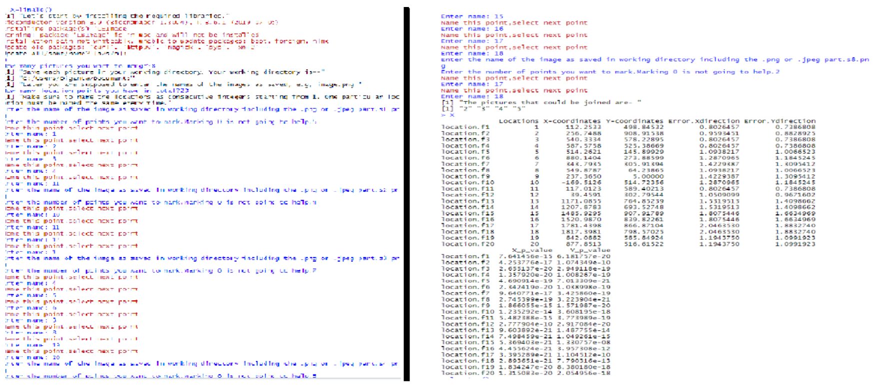


Figure 8: Original Picture

- CASE 3:ROTATIONS AND DIFFERENT RESOLUTION.
  The data set can be found here- https://www.isical.ac.in/ arnabc/linmod/projects.html



```
> x=limits()
[1] "Let's start by installing the required libraries."
Bioconductor version 3.9 (BiocManager 1.30.4), R 3.6.1 (2019-07-05)
Installing package(s) 'EBImage'
Warning: package 'EBImage' is in use and will not be installed
Installation path not writeable, unable to update packages: boot, foreign, nlme
Update old packages: 'curl', 'httpuv', 'magick', 'sys', 'xml2'
Update all/some/none? [a/s/n]:
n
How many pictures you want to merge?:8
[1] "Save each picture in your working directory. Your working directory is--"
[1] "C:/Users/Diganta/Documents"
[1] "Later you are supposed to enter the names of the images as saved, e.g, image.png "
How many location points you have in total?20
[1] "Make sure to name the locations as consecutive integers starting from 1. One particular loc
ation must be named the same every time."
Enter the name of the image as saved in working directory including the .png or .jpeg part.s1.pn
g
Enter the number of points you want to mark.Marking 0 is not going to help.5
Name this point,select next point
Enter name: 1
Name this point,select next point
Enter name: 2
Name this point,select next point
Enter name: 3
Name this point,select next point
Enter name: 4
Name this point,select next point
Enter name: 11
Enter the name of the image as saved in working directory including the .png or .jpeg part.s2.pn
g
Enter the number of points you want to mark.Marking 0 is not going to help.4
Name this point,select next point
Enter name: 10
Name this point,select next point
Enter name: 11
Name this point,select next point
Enter name: 12
Name this point,select next point
Enter name: 1
Enter the name of the image as saved in working directory including the .png or .jpeg part.s3.pn
g
Enter the number of points you want to mark.Marking 0 is not going to help.7
Name this point,select next point
Enter name: 4
Name this point,select next point
Enter name: 5
Name this point,select next point
Enter name: 6
Name this point,select next point
Enter name: 3
Name this point,select next point
Enter name: 8
Name this point,select next point
Enter name: 19
Name this point,select next point
Enter name: 20
Enter the name of the image as saved in working directory including the .png or .jpeg part.s4.pn
g
Enter the number of points you want to mark.Marking 0 is not going to help.5
```

```
Enter name: 15
Name this point,select next point
Enter name: 16
Name this point,select next point
Enter name: 17
Name this point,select next point
Enter name: 18
Enter the name of the image as saved in working directory including the .png or .jpeg part.s8.pn
g
Enter the number of points you want to mark.Marking 0 is not going to help.2
Name this point,select next point
Enter name: 17
Name this point,select next point
Enter name: 18
[1] "The pictures that could be joined are- "
[1] "2" "3" "4" "5"
> x
             Locations X-coordinates Y-coordinates Error.Xdirection Error.Ydirection
location.f1          1      112.2533     498.84532        0.8026457        0.7386808
location.f2          2      256.7488     908.91538        0.9593451        0.8828925
location.f3          3      540.3334     578.22895        0.8026457        0.7386808
location.f4          4      587.5758     525.38669        0.8026457        0.7386808
location.f5          5      514.2621     145.89929        1.0938217        1.0066523
location.f6          6      880.1404     273.88599        1.2870965        1.1845245
location.f7          7      343.7935     305.91394        1.4229387        1.3095412
location.f8          8      549.8787      64.23865        1.0938217        1.0066523
location.f9          9      237.3650       5.00000        1.4229387        1.3095412
location.f10        10     -169.5126     514.72356        1.2870965        1.1845245
location.f11        11      117.0123     589.40213        0.8026457        0.7386808
location.f12        12       39.4591     302.79544        1.0509099        0.9671602
location.f13        13     1171.0855     764.85239        1.5319513        1.4098662
location.f14        14     1207.8783     693.52748        1.5319513        1.4098662
location.f15        15     1485.9295     907.91789        1.8075446        1.6634969
location.f16        16     1520.9870     839.82261        1.8075446        1.6634969
location.f17        17     1781.4398     866.87104        2.0463530        1.8832740
location.f18        18     1817.3981     798.57025        2.0463530        1.8832740
location.f19        19      842.0882     585.84924        1.1943750        1.0991923
location.f20        20      877.8513     516.61522        1.1943750        1.0991923
                X_p_value    Y_p_value
location.f1  7.641456e-15 6.181757e-20
location.f2  4.253776e-17 1.074349e-10
location.f3  2.655137e-20 2.949118e-19
location.f4  1.357920e-20 1.008267e-19
location.f5  4.690914e-19 7.013309e-21
location.f6  2.342419e-20 1.038998e-19
location.f7  9.640771e-17 3.425860e-19
location.f8  2.745399e-19 3.223904e-21
location.f9  1.866055e-15 1.571987e-20
location.f10 1.235292e-14 3.608195e-18
location.f11 5.482388e-15 3.773989e-19
location.f12 2.777904e-10 2.917084e-20
location.f13 9.603892e-21 1.487755e-14
location.f14 7.498459e-21 1.049261e-15
location.f15 5.369403e-21 1.330757e-08
location.f16 4.455624e-21 3.957308e-12
location.f17 3.395289e-21 1.104512e-10
location.f18 2.893651e-21 7.790316e-13
location.f19 1.834247e-20 8.380180e-18
location.f20 1.315083e-20 2.054956e-18
```

Figure 9: Final Locations