

A Landmark-based Framework for Image Comparison in the Presence of Various Geometric Misalignments - Local Change Detection

Anik Roy, Sagar Ghosh, Partha Sarathi Mukherjee

2026-03-28

Description: This code snippet intakes two image paths and outputs the position on the second image where the maximal change has happened compared to the first image

```
#if (!requireNamespace("BiocManager", quietly = TRUE))
#  install.packages("BiocManager")

#BiocManager::install("EBImage")

library(EBImage)
#library(raster)
library(ggplot2)
library(reshape2)
library(sp)
library(magick)
library(OpenImageR)
library(foreach)
library(doParallel)
library(DRIP)
library(twosamples)
library(OpenImageR)
library(magic)
library(pracma)
library(jpeg)
library(SAFARI)
#library(imager)

local_change_detection<- function(image_1_path, image_2_path){
  ## Helper Functions ##

  # 1. Euclidean Invariant Metric

  Centering_matrix=function(k){
    C=matrix(0,nrow=k,ncol = k)
    for(i in 1:k){
      for(j in 1:k){
        if(i==j){
          C[i,j]=(k-1)/k
        }
      }
    }
  }
}
```

```

        else{
            C[i,j]=(-1)/k
        }
    }
}
return(C)
}

# 2. Frobenious Norm

frob_norm=function(mat){
    a=nrow(mat)
    b=ncol(mat)
    d=0
    for(i in 1:a){
        for(j in 1:b){
            d=d+mat[i,j]^2
        }
    }
    return(sqrt(d))
}

# 3. TRS invariant metric

TRS=function(X1,X2){
    k=nrow(X1)
    C=Centering_matrix(k)
    Z1=(C%*%X1)
    Z2=(C%*%X2)
    T1=(Z1%*%t(Z1))
    T2=(Z2%*%t(Z2))
    dist_matrix=(1/(frob_norm(T1)))*T1 - (1/(frob_norm(T2)))*T2
    dist=max(abs((dist_matrix)))
    return(dist)
}

# 3.1 TRS Local Change Detection
TRS_local <- function(X1, X2) {
    k <- nrow(X1)
    C <- Centering_matrix(k)

    Z1 <- C %*% X1
    Z2 <- C %*% X2

    T1 <- Z1 %*% t(Z1)
    T2 <- Z2 %*% t(Z2)

    dist_matrix <- (1 / frob_norm(T1)) * T1 - (1 / frob_norm(T2)) * T2

    # Maximum absolute change
    dist <- max(abs(dist_matrix))

    # Indices where maximum change occurs

```

```

idx <- which(abs(dist_matrix) == dist, arr.ind = TRUE)

# Indices of X1 involved (row/column indices map directly to rows of X1)
X1_indices <- unique(as.vector(idx))

return(list(
  dist = dist,
  dist_matrix = dist_matrix,
  max_change_indices = idx,      # (i, j) pairs in the matrix
  X1_indices = X1_indices        # which rows of X1 correspond
))
}

```

4. Boundary curve estimation

```

sample_edge=function(X,h_n){
  #X: Image matrix
  #X=readJPEG("2004-1c1.jpg")[, ,1]
  est_img=JPLLK_surface(X,3,plot = F)$fitted
  e=stepEdge(est_img,h_n,(qnorm(0.999,0,1)*JPLLK_surface(est_img,3)$sigma)
    ,degree=0,plot = F)

  e[1,]=0
  e[2,]=0
  e[,1]=0
  e[,2]=0
  e[nrow(X),]=0
  e[nrow(X)-1,]=0
  e[,ncol(X)]=0
  e[,ncol(X)-1]=0
  for(k in 1:nrow(X)){
    for(l in 1:ncol(X)){
      if(k>2 && l>2 && k< (nrow(X)-1) && l< (ncol(X)-1) && e[k,l]==1 &&
        sum(e[(k-1):(k+1),(l-1):(l+1)])<=4){
        e[k,l]=0
      }
    }
  }

  e[1,]=0
  e[2,]=0
  e[,1]=0
  e[,2]=0
  e[nrow(X),]=0
  e[nrow(X)-1,]=0
  e[,ncol(X)]=0
  e[,ncol(X)-1]=0
  for(k in 1:nrow(X)){
    for(l in 1:ncol(X)){
      if(k>2 && l>2 && k< (nrow(X)-1) && l< (ncol(X)-1) && e[k,l]==1 &&
        sum(e[(k-1):(k+1),(l-1):(l+1)])<=3){
        e[k,l]=0
      }
    }
  }
}

```

```

    }
  }
  e[,1]=0
  e[,2]=0
  e[,1]=0
  e[,2]=0
  e[nrow(X),]=0
  e[nrow(X)-1,]=0
  e[,ncol(X)]=0
  e[,ncol(X)-1]=0
  for(k in 1:nrow(X)){
    for(l in 1:ncol(X)){
      if(k>2 && l>2 && k< (nrow(X)-1) && l< (ncol(X)-1) && e[k,l]==1 &&
        sum(e[(k-1):(k+1),(l-1):(l+1)])<=3){
        e[k,l]=0
      }
    }
  }
  }
  P=as.matrix(which(e==1,arr.ind = T))
  #plot(P)
  return(P)
}

#5. Landmark Point Selection

choose_landmarks <- function(P,
                             center = NULL,
                             K = NULL,
                             dtheta = NULL,
                             angle_start = 0,
                             tol = 1.0,
                             max_tol = 5.0) {
  stopifnot(is.matrix(P) || is.data.frame(P))
  P <- as.matrix(P)
  stopifnot(ncol(P) == 2)
  colnames(P) <- c("x", "y")

  if (is.null(center)) center <- colMeans(P)
  stopifnot(length(center) == 2, is.finite(center[1]), is.finite(center[2]))

  if (is.null(K) && is.null(dtheta))
    stop("Provide either K or dtheta (degrees).")
  if (!is.null(dtheta)) {
    stopifnot(dtheta > 0)
    dtheta_rad <- dtheta * pi / 180
    K <- max(1L, round(2 * pi / dtheta_rad))
  }
  stopifnot(K >= 1)

  # Anticlockwise target angles
  angles <- angle_start + 2 * pi * (0:(K - 1)) / K

```

```

# Vectors from center to each edge point
V <- sweep(P, 2, center, FUN = "-")      # n x 2
n <- nrow(P)

pick_for_angle <- function(theta) {
  # Direction of the ray
  d <- c(cos(theta), sin(theta))
  # Perpendicular (left normal) to measure distance from the ray
  nperp <- c(-sin(theta), cos(theta))

  # Longitudinal (along-ray) and lateral (perpendicular) components
  t <- V %*% d      # projection length along the ray
  s <- V %*% nperp   # signed perpendicular distance to the ray

  # Expand tolerance until we find candidates, preferring the farthest along the ray
  tol_k <- tol
  chosen <- NA_integer_
  while (is.na(chosen) && tol_k <= max_tol) {
    cand <- which(abs(s) <= tol_k & t > 0)
    if (length(cand) > 0) {
      chosen <- cand[which.max(t[cand])] # farthest hit along this ray
    } else {
      tol_k <- tol_k * 1.5
    }
  }

  # If still nothing, fall back to nearest-angle point
  # (dense edges usually make this rare)
  if (is.na(chosen)) {
    phi <- atan2(V[, 2], V[, 1])
    adiff <- abs(atan2(sin(phi - theta), cos(phi - theta))) # circular difference
    chosen <- which.min(adiff)
  }
  chosen
}

idx <- vapply(angles, pick_for_angle, integer(1))
L <- P[idx, , drop = FALSE]
rownames(L) <- NULL
attr(L, "center") <- center
attr(L, "angles") <- angles
attr(L, "indices") <- idx
L
}

plot_landmarks <- function(P, L, center = attr(L, "center")) {
  plot(P, asp = 1, pch = 16, cex = 0.4, col = "grey70",
       xlab = "x", ylab = "y", main = "Landmarks by Ray Casting")
  points(L, pch = 19)
  segments(center[1], center[2], L[, 1], L[, 2], lty = 3)
  points(center[1], center[2], pch = 4, lwd = 2, cex = 1.2)
  text(L, labels = seq_len(nrow(L)), pos = 3, cex = 0.7)
}

```

```

# The Null Image

image_1=image_read(image_1_path)

im_1=image_resize(image_1,"256x256")

im_1_gray=image_convert(im_1, colorspace = 'gray')
#im_1_negative=image_negate(im_1_gray)
n1=dim(image_data(im_1_gray))[2]
n2=dim(image_data(im_1_gray))[3]
im_1_matrix=matrix(as.numeric(image_data(im_1_gray)[1,,])*(1/256), nrow=n1,ncol=n2)

#Landmark matrix construction for the first image
x_1=matrix(0,n1,n2,dimnames=list(1:n1,1:n2))
for(i in 1:n1){
  for(j in 1:n2){
    x_1[i,j]=im_1_matrix[i,j]
  }
}

Im_1=x_1+matrix(rnorm(nrow(x_1)*ncol(x_1),0,0.01),nrow=nrow(x_1),ncol=ncol(x_1))
#image(Im_1,useRaster=TRUE,axes=FALSE,col = gray(0:256 / 256))

P1=sample_edge(Im_1, 2)
L1=choose_landmarks(P1, dtheta=1)
L1_mat <- as.matrix(L1)*(1/256)


#The Distorted Image
image_2=image_read(image_2_path)

im_2=image_resize(image_2,"256x256")

im_2_gray=image_convert(im_2, colorspace = 'gray')
#im_1_negative=image_negate(im_1_gray)
n1=dim(image_data(im_2_gray))[2]
n2=dim(image_data(im_2_gray))[3]
im_2_matrix=matrix(as.numeric(image_data(im_2_gray)[1,,])*(1/256), nrow=n1,ncol=n2)

#Landmark matrix construction for the first image
x_2=matrix(0,n1,n2,dimnames=list(1:n1,1:n2))
for(i in 1:n1){
  for(j in 1:n2){
    x_2[i,j]=im_2_matrix[i,j]
  }
}

Im_2=x_2+matrix(rnorm(nrow(x_2)*ncol(x_2),0,0.01),nrow=nrow(x_2),ncol=ncol(x_2))
#image(Im_2,useRaster=TRUE,axes=FALSE,col = gray(0:256 / 256))

P2=sample_edge(Im_2, 2)

```

```

L2=choose_landmarks(P2, dtheta=1)
L2_mat <- as.matrix(L2)*(1/256)

# Detecting where the maximum changes have happened:
Ch <- TRS_local(L1_mat, L2_mat)$X1_indices

library(ggplot2)
library(reshape2)

# ----- Prepare image data for ggplot -----
img_df <- melt(Im_2)
colnames(img_df) <- c("x", "y", "intensity")
img_df$y <- img_df$y # flip y-axis for correct orientation

# Maximum-change points
X1_indices <- Ch
cross_size <- 5 # Define the size of the cross segments
radius <- 10

# Extract coordinates
points_df <- data.frame(
  x = L2[X1_indices, 1],
  y = L2[X1_indices, 2] # flip y for plotting
)

# Function to generate circle points
circle_points <- function(cx, cy, r, n = 100) {
  theta <- seq(0, 2*pi, length.out = n)
  data.frame(
    x = cx + r * cos(theta),
    y = cy + r * sin(theta)
  )
}

# Create circles data frame
circles_df <- do.call(rbind, lapply(1:nrow(points_df), function(i) {
  cbind(circle_points(points_df$x[i], points_df$y[i], radius), id = i)
}))

# Plot
ggplot() +
  # Base image
  geom_raster(data = img_df, aes(x = x, y = y, fill = intensity)) +
  scale_fill_gradient(low = "black", high = "white") +

  # Crosses (Now correctly placed on top of the raster)
  geom_segment(data = points_df, aes(x = x - cross_size, xend = x + cross_size,
                                     y = y, yend = y),
              , color = "red", size = 1.0) + # Decreased size slightly for cleaner look
  geom_segment(data = points_df, aes(x = x, xend = x,

```

```

        y = y - cross_size, yend = y + cross_size)
    , color = "red", size = 1.0) + # Decreased size slightly for cleaner look

# Circles as paths (Drawn around the cross)
geom_path(data = circles_df, aes(x = x, y = y, group = id), color = "red", size = 1.5) +

theme_minimal() +
theme(axis.title = element_blank(),
      axis.text = element_blank(),
      axis.ticks = element_blank(),
      legend.position = "none") +
coord_fixed()

}

image_1_path = "~/Documents/Stat_Project_PSM/Lake_Images/2014-3c.jpg"
image_2_path = "~/Documents/Stat_Project_PSM/Lake_Images/2021-4ac.jpg"

local_change_detection(image_1_path, image_2_path)

```

