

A Landmark-based Framework for Image Comparison in the Presence of Various Geometric Misalignments - Power Computation Polygon

Anik Roy, Sagar Ghosh, Partha Sarathi Mukherjee

2026-03-28

Description: This code snippet intakes two Polygon images and outputs the power of the test described in Section 4.2 of the manuscript, where the test treats the first image as null (H_0) and the second image as alternate (H_1). More precisely, this particular demonstration computes the power of the test $g_{P,0}$ vs $g_{P,2}^{(H_1)}$.

```
#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)
```

```
# 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)
}

```

Null and Alternate Images

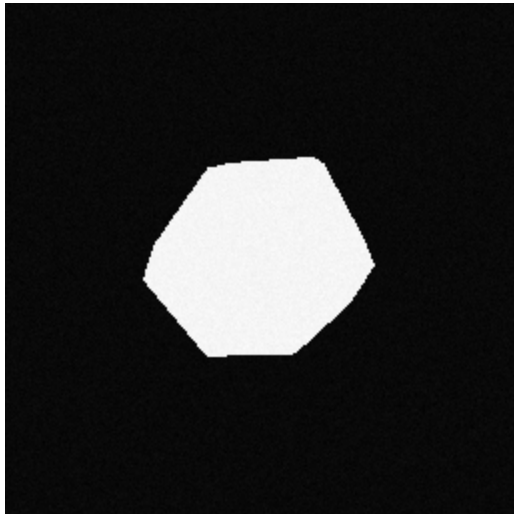
Null Ellipse: $g_{P,0}$, Alternate Ellipse: $g_{P,2}^{(H_1)}$

```
convex_polygon_image <- function(m=300, n=300, R=0.9, H=64) {
  set.seed(7)
  th <- runif(H, 0, 2*pi)
  # normals
  N <- cbind(cos(th), sin(th))
  # random radii around R (keep positive & bounded to remain convex and closed)
  r <- R * (0.7 + 0.3*runif(H))
  b <- r # since support in direction N_k is r_k

  xs <- seq(-1, 1, length.out = n)
  ys <- seq(-1, 1, length.out = m)
  XX <- matrix(rep(xs, each = m), nrow = m)
  YY <- matrix(rep(ys, times = n), nrow = m)

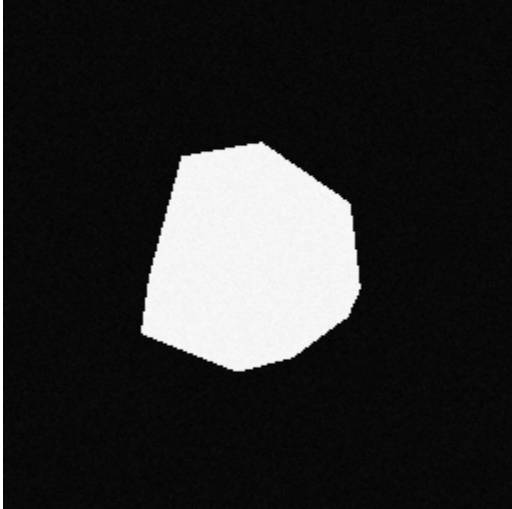
  img <- matrix(1L, m, n)
  for (k in seq_len(H)) {
    img[ (N[k,1]*XX + N[k,2]*YY) > b[k] ] <- 0L
  }
  img
}
x <- convex_polygon_image(256, 256, R=0.52, H=50)

nimg <- x + matrix(rnorm(nrow(x)*ncol(x),0,0.01),nrow=nrow(x),ncol=ncol(x))
image(nimg ,useRaster=TRUE,axes=FALSE,col = gray(0:256 / 256), asp=1)
```



```
# # Alternative image
x_1 <- translation(convex_polygon_image(256, 256, R=0.52, H=20)
  ,shift_rows = 3,shift_cols = 2)

nimg1 <- x_1 + matrix(rnorm(nrow(x)*ncol(x),0,0.01),nrow=nrow(x),ncol=ncol(x))
image(nimg1 ,useRaster=TRUE,axes=FALSE,col = gray(0:256 / 256), asp=1)
```



Empirical Power Computation

```

h=2
library(DRIP)
nimg_1 <- x+matrix(rnorm(nrow(x)*ncol(x),0,0.01),nrow=nrow(x),ncol=ncol(x))
nimg_2 <- x_1+matrix(rnorm(nrow(x)*ncol(x),0,0.01),nrow=nrow(x),ncol=ncol(x))
est_img_1 <- JPLLK_surface(x,3,plot = F)$fitted
# Jump preserving estimate of image 1
est_img_2 <- JPLLK_surface(x_1,3,plot = F)$fitted
# Jump preserving estimate of image 2
edge_1 <- stepEdge(est_img_1,h,(qnorm(0.999,0,1)*JPLLK_surface(est_img_1,3)$sigma)
,degree=0,plot = F)
edge_2 <- stepEdge(est_img_2,h,(qnorm(0.999,0,1)*JPLLK_surface(est_img_2,3)$sigma)
,degree=0,plot = F)
resid1 <- nimg_1-est_img_1
resid2 <- nimg_2-est_img_2

sigma1_hat <- JPLLK_surface(x,3,plot = F)$sigma
sigma2_hat <- JPLLK_surface(x_1,3,plot = F)$sigma

#-----Generate null distribution -----#
epoch = 100
TRS_dist_H0=array(0,epoch)
for(q in 1:epoch){
  set.seed(2025+q)
  nimg_1.star <- est_img_1 + matrix(rnorm(nrow(x)*ncol(x),0,sigma1_hat)
, nrow=nrow(x),ncol=ncol(x))
  nimg_2.star <- translation(rotateImage(est_img_1,runif(1,0,20)
, threads = 1)
, shift_rows = sample(1:10,1)
, shift_cols = sample(1:10,1))
+ matrix(rnorm(nrow(x)*ncol(x),0,sigma2_hat)
, nrow=nrow(x),ncol=ncol(x))

P1 <- sample_edge(nimg_1.star,h)
L1 <- choose_landmarks(P1, dtheta = 1)

```

```

P2 <- sample_edge(nimg_2.star,h)
L2 <- choose_landmarks(P2, dtheta = 1)
TRS_dist_H0[q] <- TRS(L1,L2)

}
cut_off <- quantile(TRS_dist_H0,0.95)

TRS_dist_H1 <- array(0,epoch)
for(q in 1:epoch){

  set.seed(2025+q)
  nimg_1.star <- est_img_2 + matrix(rnorm(nrow(x)*ncol(x),0,sigma1_hat)
                                   ,nrow=nrow(x),ncol=ncol(x))
  nimg_2.star <- translation(rotateImage(est_img_1,runif(1,0,20)
                                   , threads = 1)
                           ,shift_rows = sample(1:10,1)
                           ,shift_cols = sample(1:10,1))
  + matrix(rnorm(nrow(x)*ncol(x),0,sigma2_hat),nrow=nrow(x)
           ,ncol=ncol(x))

  P1 <- sample_edge(nimg_1.star,h)
  L1 <- choose_landmarks(P1, dtheta = 1)
  P2 <- sample_edge(nimg_2.star,h)
  L2 <- choose_landmarks(P2, dtheta = 1)
  TRS_dist_H1[q] <- TRS(L1,L2)

}
power <- sum(TRS_dist_H1>cut_off)/epoch

print(power)

## [1] 1

```