vidProcessing.R

ayanb\_000

Tue Jan 05 21:18:10 2016

Thresholds a object for green pixels

Green pixels, or pixels with H value between 61/360 and 140/360 and S value over 0.15 in the HSV color scheme, are treated as foreground. All other pixels are treated as background. This is useful in detecting the Green Flourescence Protein used to denote that a cell is alive. @param frames A object with all 3 color channels(R,G,B) @return a binary object in the green color channel @examples library(flowcatchR) threshedFrames <- greenThresh(MesenteriumSubset) @export @author Ayan Bandyopadhyay, Bellarmine College Prep 11/26/2015

greenThresh <-function(frames)  
{  
 # checks if required packages are loaded  
 if (!requireNamespace("flowcatchR", quietly = TRUE) )  
 {  
 stop("flowcatchR needed for this function to work. Please install it.", call. = FALSE)  
 }  
 threshedFrames = array(dim = c(dim(frames)[1],dim(frames)[2],frameCount))  
  
 for (i in 1:frameCount)  
 {  
 oldGreenImg= c(frames[,,2,i])  
 oldBlueImg = c(frames[,,3,i])  
 oldRedImg = c(frames[,,1,i])  
 # createGreenImage, createRedImage, and createBlueImage turn gray pixels into background pixels  
 createGreenImage <- function(oldGreen,oldRed,oldBlue)  
 {  
 if(oldRed == oldBlue && oldBlue == oldGreen)  
 {  
 oldGreen=0  
 }  
 return(oldGreen)  
 }  
 createRedImage <- function(oldGreen,oldRed,oldBlue)  
 {  
 if(oldRed == oldBlue && oldBlue == oldGreen)  
 {  
 oldRed=0  
 }  
 return(oldRed)  
 }  
 createBlueImage <- function(oldGreen,oldRed,oldBlue)  
 {  
 if(oldRed == oldBlue && oldBlue == oldGreen)  
 {  
 oldBlue=0  
 }  
 return(oldBlue)  
 }  
 r <-mapply(createRedImage,oldGreenFirst,oldRedFirst,oldBlueFirst)  
 g <-mapply(createGreenImage,oldGreenFirst,oldRedFirst,oldBlueFirst)  
 b <-mapply(createBlueImage,oldGreenFirst,oldRedFirst,oldBlueFirst)  
 r <- matrix(r,nrow = dim(frames)[1], ncol = dim(frames)[2])  
 g <- matrix(g,nrow = dim(frames)[1], ncol = dim(frames)[2])  
 b <- matrix(b,nrow = dim(frames)[1], ncol = dim(frames)[2])  
 r <- c(r)  
 g<- c(g)  
 b <- c(b)  
  
 hsvColors <- rgb2hsv(r=r,g=g,b=b)  
 imgH <- hsvColors[1,]  
 imgS <- hsvColors[2,]  
 imgV <- hsvColors[3,]  
 grayscaleVector <- vector(length=dim(frames)[1]\*dim(frames)[2])  
  
 # createBinaryVector creates a pixel of value 1 if hue is green and saturation>0.15  
 createBinaryVector <- function(grayVector,h,s,v)  
 {  
  
 if (h>(61/360) && h<(140/360) && s>0.15)  
 {  
 grayVector = 1  
 }  
 else  
 {  
 grayVector = 0  
 }  
 return(grayVector)  
 }  
  
 grayscaleVector <- mapply(createBinaryVector,grayVector=grayscaleVector,h=imgH,s=imgS,v=imgV)  
  
 # binaryImage holds the thresholded image. Green pixels are white, the rest is black.  
 binaryImage <- matrix(grayscaleVector, ncol = dim(frames)[2],nrow = dim(frames)[1])  
 threshedFrames[,,i] <- binaryImage  
 }  
 rgbFrames = EBImage::channel(as.Image(threshedFrames),'rgb')  
 greenFrames = channel.Frames(rgbFrames, "green")  
 return(greenFrames)  
}

Generate a object with corrected y-values

Returns a TrajectorySet so that the y value is the distance from the bottom of the Frames object, not from the top @param particles A object @param L Maximum number of pixels an object can move in two consecutive frames @param R Linkrange, i.e. the number of consecutive frames to search for potential candidate links @param epsilon1 A numeric value, to be used in the formula. Jitter for allowing angular displacements @param epsilon2 A numeric value, to be used in the formula. Jitter for allowing spatial displacements @param lambda1 A numeric value. Multiplicative factor for the penalty function @param lambda2 A numeric value. Multiplicative factor applied to the angular displacement @param penaltyFunction A function structured in such a way to be applied as penalty function in the linking @param verboseOutput Logical, whether the output should report additional intermediate steps. For debugging use mainly. @param prog Logical, whether the a progress bar should be shown during the tracking phase @param include.intensity Logical, whether to include also intensity change of the particles in the cost function calculation @param include.area Logical, whether to include also area change of the particles in the cost function calculation @param frames The object that the object is derived from @return A object @examples library(flowcatchR) platelets <-particles(channel.Frames(MesenteriumSubset,"red")) trajSet <- generateTraj(platelets, L=26, R=3, epsilon1=0, epsilon2=0, lambda1=1, lambda2=0, penaltyFunction=penaltyFunctionGenerator(), include.area=FALSE, MesenteriumSubset) @export @author Ayan Bandyopadhyay, Bellarmine College Prep 11/26/2015

generateTraj <- function (particles,  
 L=26, R=3,  
 epsilon1=0, epsilon2=0,  
 lambda1=1, lambda2=0,  
 penaltyFunction=penaltyFunctionGenerator(),  
 include.area=FALSE, frames)  
{  
 linkedParticles <- link.particles(particles,  
 L=L, R=R,  
 epsilon1=epsilon1, epsilon2=epsilon2,  
 lambda1=lambda1, lambda2=lambda2,  
 penaltyFunction=penaltyFunction,  
 verboseOutput=FALSE, prog=FALSE,  
 include.intensity=TRUE,include.area=TRUE)  
 trajParticles <- trajectories(linkedParticles)  
 # Correct trajectory data  
 for (i in 1:length(trajParticles))  
 {  
 newYCoords <- dim(frames)[2] - (trajParticles[[i]]$trajectory$yCoord)  
 trajParticles[[i]]$trajectory$yCoord <- newYCoords  
 }  
 return(trajParticles)  
}

Generates 24 measures for each trajectory

Removes trajectories with less than 4 data points @param trajSet This is a object, @return A matrix 24 columns wide. Each row corresponds to one trajectory. @examples library(flowcatchR) trajPlatelets <- trajectories(particles(channel.Frames(MesenteriumSubset,"red"))) trajData <-trajMeasures(trajPlatelets) @export @author Ayan Bandyopadhyay, Bellarmine College Prep 11/26/2015

trajMeasures <- function(trajSet)  
{  
 library(class)  
 library(traj)  
  
 # create 2 vectors: trajVector and index  
 trajData <- list()  
 index <- vector(length = length(trajSet))  
 for(i in 1:length(trajSet))  
 {  
 vec <- trajSet[[i]]$trajectory$yCoord  
 trajData[[i]]<- vec  
 index[i] <- length(vec)  
 }  
 trajVector <- as.vector(do.call("rbind", lapply(trajData, as.data.frame)))  
  
 # create matrix of 24 measurements for each trajectory  
 trajDataMatrix = matrix(ncol = 24, nrow = length(index))  
 for (i in 1:length(index))  
 {  
 if(index[i]<4)  
 {  
 trajVector <- trajVector[-(1:index[i])]  
 }  
 else  
 {  
 newVector <- trajVector[1:index[i]]  
 newFrame <- matrix(nrow=2,ncol=index[[i]],append(newVector,1:index[i]),byrow= TRUE)  
 timeFrame <- matrix(nrow=2,ncol=index[[i]],append(1:index[i],1:index[i]),byrow= TRUE)  
 s1 <- step1measures(newFrame,timeFrame, ID = FALSE)  
 trajData <- as.numeric(s1$measurments[1,])[2:length(as.vector(s1$measurments[1,]))]  
 trajDataMatrix[i,] <- trajData  
  
 if(length(trajVector) > length(1:index[i]))  
 {  
 trajVector <- trajVector[-(1:index[i])]  
 }  
 else  
 {  
 trajVector <- trajVector  
 }  
 }  
 }  
 # get rid of data for trajectories w/ less than 4 data points  
 for(i in 1:length(trajDataMatrix[,1]))  
 {  
 if( is.na((trajDataMatrix[,1])[i]) )  
 {  
 trajDataMatrix <- trajDataMatrix[-i,]  
 }  
 else  
 {  
 trajDataMatrix <- trajDataMatrix  
 }  
 }  
  
 trajDataFrame <- as.data.frame(trajDataMatrix)  
 return(trajDataFrame)  
}

Generate KNN classifier with LOOCV

Fits trajectory data into a K-nearest neighbors classifier using leave one out cross validation. @param trajDataFrame A data frame with measures for each trajectory. This can be produced by function trajMeasures @param labelVector A vector of labels for each trajectory. Its length must be equal to the number of rows in trajDataFrame @param kVal The number of neighbors used for classification @return A vector of classified labels for each trajectory @examples data <- as.data.frame(matrix(1:4,nrow = 2,ncol = 2)) labels <- c("live","dead") classifierKNN\_CV<- fitModelKNN\_CV(data,labels,3) @export @author Ayan Bandyopadhyay, Bellarmine College Prep 11/26/2015

fitModelKNN\_CV<- function (trajDataFrame, labelVector, kVal)  
{  
 dataFrame<- trajDataFrame  
 dataFrame$cellStatus <-labelVector  
 dataLabels <- dataFrame[,dim(dataLabels)[2]]  
 dataTrain <- dataFrame[,1:(dim(dataLabels)[2]-1)]  
  
 trajDataCV<- knn.cv(train = dataTrain, cl = dataLabels, k=kVal)  
 mp <- sum(trajDataCV != dataLabels)/length(dataLabels)  
 print("Misclassification probability: " + mp)  
  
 return(trajDataCV)  
}