day = read.csv("sample\_day\_filtered.csv")

light = read.csv("faulty\_light\_filtered.csv")

grip = read.csv("degrading\_grip\_filtered.csv")

lg = read.csv("faulty\_light\_degrading\_grip\_filtered.csv")

# function to calculate trends from data

gettrends <- function(data, len){

full = dim(data)[1]

trends = matrix(nrow = (full-len), ncol = 4)

x = c(1:len)

for (i in (len+1):full){

trends[(i-len),1] = data[(i-len),1]

for (j in 1:3){

model = lm(data[(i-len):(i-1), (j+1)] ~ x)

trends[(i-len),(j+1)] = model$coefficients[2]

}

}

return(trends)

}

len = 10

# get means and trends from historic data

historicmeans = colMeans(day)[2:4]

historictrends = gettrends (day, len)

# get limits on values

mlim1 = 5.0\*sqrt(var(abs(day[,2] - historicmeans[1])))

mlim2 = 5.0\*sqrt(var(abs(day[,3] - historicmeans[2])))

mlim3 = 5.0\*sqrt(var(abs(day[,4] - historicmeans[3])))

lims = c(mlim1, mlim2, mlim3)

# get change on trends and limits on change from historic data

l = dim(historictrends)[1]

change = historictrends[2:l,2:4]- historictrends[1:(l-1),2:4]

clim1 = 3.0\*sqrt(var(change[,1]))

clim2 = 3.0\*sqrt(var(change[,2]))

clim3 = 3.0\*sqrt(var(change[,3]))

changelims = c(clim1, clim2, clim3)

# function to calculate time to violation

violationtime <- function(s, vmap, trends, tstep, maxtime, msteps, printnum){

border = vmap[which(vmap[,4] == 0),1:3]

edge = vmap[which(vmap[,4] == 2),1:3]

borderedge = rbind(border, edge)

R = sqrt(trends[1]^2 + trends[2]^2 + trends[3]^2)

x = borderedge[,1]-s[1]

y = borderedge[,2]-s[2]

z = borderedge[,3]-s[3]

betype = c(rep("b", dim(border)[1]), rep("e", dim(edge)[1]))

type = "v"

timepred = 0

# before considering a move, check whether already on a border/edge point

ind0 = which((abs(x) < (msteps[1]/2)) & (abs(y) < (msteps[2]/2)) & (abs(z) < (msteps[3]/2)))

if (length(ind0) > 0) {

type = betype[ind0]

if (printnum == 1) print(c(type, "this is a boundary/edge point"))

}

else {

# don't consider border/edge points in wrong direction

indsx = which( (abs(x) < 0.00001) | (sign(x) == sign(trends[1])))

indsy = which( (abs(y) < 0.00001) |(sign(y) == sign(trends[2])))

indsz = which( (abs(z) < 0.00001) |(sign(z) == sign(trends[3])))

inds = intersect(indsx, intersect(indsy, indsz))

if (length(inds) > 0) {

x = x[inds]

y = y[inds]

z = z[inds]

keep = borderedge[inds,]

betype = betype[inds]

if (length(inds) > 1) {

B = sqrt(x^2 + y^2 + z^2)

C = (x\* trends[1] + y\*trends[2] + z\*trends[3])/(R\*B)

# deal with precision problems

C[which(C > 1)] = 1

C[which(C < -1)] = -1

angle = acos(C)

angle = (angle + 2\*pi) %% (2\*pi)

sorted = sort.int(angle, index.return = T)

# could be multiple points with the same angle

allsame = sorted$ix[which(sorted$x == sorted$x[1])]

# find the point with the shortest distance in direction of change

ix = allsame[which(B[allsame] == min(B[allsame]))]

keepx = keep[ix,1]

keepy = keep[ix,2]

keepz = keep[ix,3]

type = betype[ix]

} else {

keepx = keep[1]

keepy = keep[2]

keepz = keep[3]

type = betype[inds]

}

displacement = sqrt((keepx-s[1])\*(keepx-s[1])+ (keepy-s[2])\*(keepy-s[2])+ (keepz-s[3])\*(keepz-s[3]))

velocity = sqrt(trends[1]\*trends[1] + trends[2]\*trends[2] + trends[3]\* trends[3])/tstep

timepred = maxtime

if (velocity > 0) timepred = displacement/velocity

timepred = floor(timepred + 0.5)

}

else {

if (printnum == 1) print(c(type, "this is a violation point"))

}

}

return(list(timepred, type))

}

vmap = read.csv("violationMap.csv", header = T)

# add "len" normal datapoints to beginning of new data

start = dim(day)[1] - len+1

end = dim(day)[1]

extra = day[start:end,]

# lose jump in faulty light data

lightnew = light

lightnew[72:240,2] = lightnew[72:240,2]+0.08

#change data file here, can be light, lightnew, grip or lg

data <- light

dataplus = rbind(extra, data)

dataplus$Time = 60\*c(0:(nrow(dataplus)-1))

# function to check time from neighbouring violation map points

checknay <- function(cp, trends, tstep, keept, vmap, maxtime, n, msteps){

nt = matrix(0, nrow = ((2\*n+1)^3), ncol = 4)

ind = 1

for (j in c(-n,0,n)) {

for (k in c(-1,0,n)) {

for (l in c(-n,0,n)) {

nt[ind, 1] = j

nt[ind, 2] = k

nt[ind, 3] = l

np = c((cp[1] + j\*msteps[1]), (cp[2] + k\*msteps[2]), (cp[3] + l\*msteps[3]))

nt[ind, 4] = violationtime(np, vmap, trends, tstep, maxtime, msteps, 2)[[1]]

ind = ind + 1

}

}

}

s = sort.int(nt[,4], index.return = T, decreasing = TRUE)

ntsorted = nt[s$ix,]

return(ntsorted[which(ntsorted[,4] >= (keept + timestep)),])

}

# function to predict violations

checktrends <- function(data, vmap, tstep, hmeans, hlims, changelims, len, maxtime, mintime, msteps){

full\_length = dim(data)[1]

# assume starting trends are all zero

keeptrends = rep(0, 3)

trends = rep(0, 3)

trendchanges = rep(0, 3)

keept = maxtime

time = data[,1]

x = c(1:len)

problem = 0

output = rep(0,10)

for (i in (len+1):full\_length){

# check whether absolute difference between actual and expected values exceed limits

if ((abs(data[i, 2] - hmeans[1]) > hlims[1]) | (abs(data[i, 3] - hmeans[2]) > hlims[2])

|(abs(data[i, 4] - hmeans[3]) > hlims[3])) {

# get trends and trend changes

for (j in 1:3){

model = lm(data[(i-len):(i-1), (j+1)] ~ x)

trends[j] = model$coefficients[2]

trendchanges[j] = abs(trends[j] - keeptrends[j])

}

change = rep("none",3)

# which trend(s) change

if (abs(data[i, 2] - hmeans[1]) > hlims[1]) change[1] = "lighting "

if (abs(data[i, 3] - hmeans[2]) > hlims[2]) change[2] = "floor friction "

if (abs(data[i, 4] - hmeans[3]) > hlims[3]) change[3] = "gripper "

# check whether this is a continuing trend

     if ((trendchanges[1] < changelims[1]) & (abs(trendchanges[2]) < changelims[2]) &(abs(trendchanges[3]) < changelims[3])) {

keept = keept-tstep

print(c(time[i], "same trends", "predicted violation in ", keept, "seconds." ))

output = rbind(output, c(data[i,1:4], "same trend", keept, change, " "))

if ((keept - timestep) < mintime){

# need to respond now, so check neighbours

nay = checknay(cp, trends, tstep, keept, vmap, maxtime, 1, msteps)

if (nrow(nay) > 0) {

for (j in 1:nrow(nay)) { print(nay[j,]) }

}

else { print("no adaptation possible.") }

}

}

else {

# calculate time to violation/edge of parameter space from current point

cp = c(data[i, 2], data[i, 3], data[i, 4])

# print(c(cp, trends))

t = violationtime(cp, vmap, trends, tstep, maxtime, msteps, 1)

if (t[[1]] < maxtime) {

keept = t[[1]]

phrase = "predicted violation in "

if (t[[2]] == "e") { phrase = "Could reach unknown parameter space in "}

print(c(time[i], " new trend", phrase, keept, "seconds." ))

output = rbind(output, c(data[i,1:4], "new trend", keept, change, t[[2]]))

keeptrends = trends

if ((keept - timestep) < mintime){

# need to respond now, so check neighbours

nay = checknay(cp, trends, tstep, keept, vmap, maxtime, 1, msteps)

if (nrow(nay) > 0) {

for (j in 1:nrow(nay)) { print(nay[j,]) }

}

else { print("no adaptation possible.") }

}

}

else { print(c(time[i], "new trend providing maximum time" , keept)) }

}

}

else {

# otherwise all ok

print(c(time[i], "No problem" ))

}

}

return(output[-1,])

}

# monitoring timestep is currently the same for all measurements

timestep = 60

# maximum time possible

maxtime = 6000000

# get msteps from violation map calculation

msteps = c(0.0125, 0.0062, 0.0125)

# minimum time allowed to predicted violation

mintime = 600

out = checktrends(dataplus, vmap, timestep, historicmeans, lims, changelims, len, maxtime, mintime, msteps)

write.csv(out, "outputlg.csv", row.names = F)

R1 = 0.6

R2 = 100

R3 = 0.35

# check a data file for violations

violationCheck<- function(data, R1, R2, R3){

for (i in 1:dim(data)[1]){

M1 = data[i, 2]

M2 = data[i, 3]

M3 = data[i, 4]

P1 = 0.2018 + (0.8191 \* M1)

P2 = 0.414 + (0.4612 \* M1)

P3 = 0.1

P4 = -0.1618+ (1.2523 \* M3)

PR = 0.8

T1 = 0

T2 = 19.765 + (15.627 \* M1)

T3 = 63.15

T1F = 0

T2F = 7.343 + ( 72.234 \* M1) + (161.485 \* M2) - (231.337 \* M1\*M2)

TR = 6.498 - (5.482 \* M1) - (6.855 \* M3) + (8.301 \* M1\*M3)

r1 = (P1\*P2\*P4) / (1 - PR\*P3)

r2 = (P1\*(P3\*T2 - P2\*T3 - P3\*T2F - T1 - T2 + T1F) + PR\*P3\*(T1F - P1\*TR - P1\*T1F) - T1F) / (PR\*P3 - 1)

r3 = (P1\*P2\*(1 - P4)) / (1 - PR\*P3)

if ((1 - PR\*P3) == 0) {

print("fail")

}

else {

if (r1 < R1){

print(c(data[i,1], "R1 fail"))

}

if (r2 > R2) {

print(c(data[i,1], "R2 fail"))

}

if (r3 > R3) {

print(c(data[i,1], "R3 fail"))

}

}

}

}