Microfauna Swimming Behavior Analysis Code

NOTE:

Where there is `R code` to follow, change anything in **`red, bold text`** to match your data.

Anything in **`blue, bold text`** can be changed, but the rest of the code must also be changed to match.

Input anything in `plain text` exactly as written.

CONTENT:

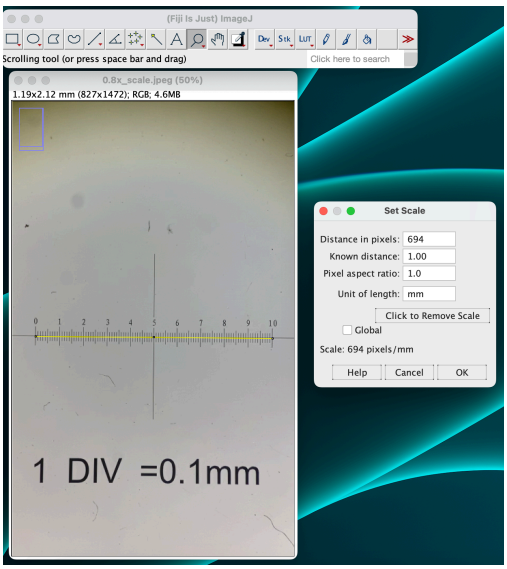
Calibration of ToxTrac Files

Prior to calibrating the instantaneous speed ("*Instant_Speed_1.txt*"), instantaneous acceleration ("*Instant_Accel_1.txt*"), and tracking coordinates ("*Tracking_RealSpace_1.txt*") files in R, obtain the pixel to mm conversion factor in ImageJ.

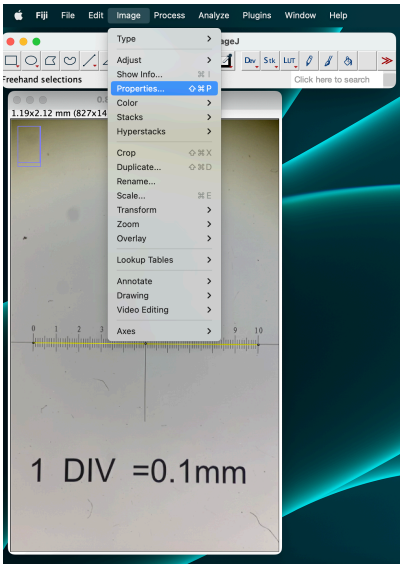
Each time recordings are made at the microscope, capture a one-second video of a micrometer slide at the same focus and magnification as used in the experiment. Use this video to create a still image of the calibration slide. If using an iPhone for recording, refrain from switching between photo and video on the phone as the camera focus and other phone settings may change.

To obtain the pixel-to-mm conversion factor:

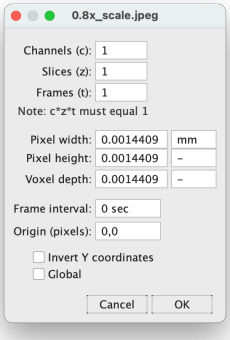
1. Open ImageJ/FIJI
2. "Set Scale" using Calibration Slide (pixels/mm)
    a. Using the "Straight Line" tool, draw a straight line using the scale bar on the micrometer (yellow line in image below).
    b. Select: Analyze > Set Scale...
    c. Adjust "Known distance" and "Unit of length" to match the distance covered by the straight line in the correct unit of measurement.
    d. Select "OK"
3. Select: Image > Properties
4. Make note of the value in in "Pixel width" and "Pixel height", this will be the pixel-to-mm conversion factor



Steps 1 and 2



Step 3



Step 4

R Code for Calibration:

1. Set working directory to file containing video *Instant_Speed_1.txt*, *Instant_Accel_1.txt*, and *Tracking_RealSpace_1.txt* files.

   In toolbar: Session > Set Working Directory > Choose Directory

2. Import files to be calibrated.

```
Instant_Speed_1 = read.delim(file="Instant_Speed_1.txt", header=TRUE, sep="\t")
Instant_Accel_1 = read.delim(file="Instant_Accel_1.txt", header=TRUE, sep="\t")
Tracking_RealSpace_1 = read.delim(file="Tracking_RealSpace_1.txt", header=TRUE,
sep="\t")
```

3. Multiply speed, acceleration, and coordinates by pixel-to-mm conversion factor (CF) obtained from ImageJ/FIJI.

```
Instant_Speed_1$Current.Speed.Calibrated=Instant_Speed_1$Current.Speed..mm.sec.* CF
Instant_Accel_1$Current.Accel.Calibrated=Instant_Accel_1$Current.Accel..mm.s.2.* CF
Tracking_RealSpace_1$Pos.X.Calibrated=Tracking_RealSpace_1$Pos..X..mm.* CF
Tracking_RealSpace_1$Pos.Y.Calibrated=Tracking_RealSpace_1$Pos..Y..mm.* CF
```

4. Export files as .csv (can be opened and used in Excel). Although they have the same name, this will not overwrite the original file, which is in .txt format.

```
write.csv(Instant_Speed_1, file='Instant_Speed_1.csv')
write.csv(Instant_Accel_1, file='Instant_Accel_1.csv')
write.csv(Tracking_RealSpace_1, file='Tracking_RealSpace_1.csv')
```

Acquiring Swimming Behavior Metrics

1. Install and load required libraries to conduct analysis of swimming behavior metrics and statistical tests.

```
library("trajr") #swimming behavior metrics
library("tidyverse") #contains ggplot2, dplyr, tidyr, readr, purrr, tibble, stringr, and forcats
library("ggpubr") #plot significance on ggplot
library("psych") #descriptive stats
library("rstatix") #Shapiro-Wilk test
library("pgirmess") #Krystal-Wallis multiple comparisons test
library(EnvStats) #Rosner outliers' test
```

2. Run the following code to acquire swimming speed and behavior metrics.

#Upload tracking coordinates file with calibration

```
Tracking_RealSpace_1 = read.csv(file="Tracking_RealSpace_1.csv", header=TRUE)
```

#Create an empty matrix. Change number of columns if adding or removing any metrics.

```
Trajr_Data = matrix(ncol = 12)
```

#Relabel column names in empty matrix. Edit list if adding or removing any metrics.

```
colnames(Trajr_Data) = c("Straightness", "Sinuosity", "Emax", "MeanDC", "SDDC",
"Turning.Angle.Mean", "Turning.Angle.Max", "Direction.Ratio", "Distance", "Length",
"Zero.Movement", "Moving.Percentage")
```

#Run loop for each unique track within the recorded video. Add or remove any metrics in the following code to match the list above.

```
for (i in unique(Tracking_RealSpace_1$Track)) {

      Track = subset(Tracking_RealSpace_1, Track == i)

      Track_xy = Track[,c(8,9,2)]

      Track_xy$Time..sec.[1] = 0

      trj = TrajFromCoords(Track_xy, xCol = 1, yCol = 2, timeCol = 3, fps = 30,
      spatialUnits = "mm", timeUnits = "s")
```

```
trj_smoothed = TrajSmoothSG(trj, p = 3, n = 31)
```

#can adjust smoothing based on personal preferences

```
Straightness = TrajStraightness(trj_smoothed)
```

#value of 1.0 = straight

```
Sinuosity = TrajSinuosity2(trj_smoothed, compass.direction = NULL)
```

#value of 0.0 = straight

```
Emax = TrajEmax(trj_smoothed, eMaxB = FALSE, compass.direction = NULL)
```

#value of 0.0 = sinuous

```
trj_DC = TrajDirectionalChange(trj_smoothed, nFrames = 1)

MeanDC = mean(trj_DC)
```

#value of 0.0 = linear

```
SDDC = sd(trj_DC)
```

#value of 0.0 = regular

```
trj_TA = TrajAngles(trj_smoothed, lag = 1, compass.direction = NULL)

Turning.Angle.Mean = mean(trj_TA)

Turning.Angle.Max = max(abs(trj_TA)) #regardless of sign (+/-)

Direction.Ratio = (length(trj_TA[trj_TA>0])/length(trj_TA[trj_TA<0]))
```

#value greater than 1 = more clockwise

#value less than 1 = more counterclockwise

```
Distance = TrajDistance(trj_smoothed, startIndex = 1, endIndex =
nrow(trj_smoothed))

Length = TrajLength(trj_smoothed, startIndex = 1, endIndex =
nrow(trj_smoothed))

trj_0M = TrajSpeedIntervals(
             trj_smoothed,
             fasterThan = -0.01,
             slowerThan = 0.01,
             interpolateTimes = TRUE,
             diff = c("backward", "central", "forward"))
```

##adjust the “fasterThan/slowerThan” speeds based on your recorded videos

```
Zero.Movement = sum(trj_0M$duration)

Moving.Percentage = ((30-Zero.Movement)/30)*100
```

```
            ##adjust "30" to match time length of video recorded


        #Compile all data into one dataframe
        Compiled = data.frame(Straightness, Sinuosity, Emax, MeanDC, SDDC,
        Turning.Angle.Mean, Turning.Angle.Max, Direction.Ratio, Distance, Length,
        Zero.Movement, Moving.Percentage)
            #Make sure the names listed match what is in the empty matrix created earlier
        #Save as a dataframe
        Trajr_Data = rbind(Trajr_Data, Compiled)
}
#End of loop


#Remove empty rows
Trajr_Data = Trajr_Data[-1,]


#Reset row names
rownames(Trajr_Data) = NULL


#Upload calibrated speed and acceleration files
Instant_Speed_1 = read.csv(file="Instant_Speed_1.csv", header=TRUE)
Instant_Accel_1 = read.csv(file="Instant_Accel_1.csv", header=TRUE)


#Create empty matrices
Speed = matrix(ncol = 2)
Acceleration = matrix(ncol = 2)


#Relabel column names
colnames(Speed) = c("Speed.Mean", "Speed.Max")
colnames(Acceleration) = c("Acceleration.Mean", "Acceleration.Max")


#Speed loop for each unique track
for (i in unique(Instant_Speed_1$Track)) {
```

```
	Speed.Mean =
	mean(Instant_Speed_1$Current.Speed.Calibrated[Instant_Speed_1$Track == i])

	Speed.Max = max(Instant_Speed_1$Current.Speed.Calibrated[Instant_Speed_1$Track
	== i])

	#Compile all data

	Compiled = data.frame(Speed.Mean, Speed.Max)

	Speed = rbind(Speed, Compiled)

}

#End of loop


#Acceleration loop for each unique track

for (i in unique(Instant_Accel_1$Track)) {

	Acceleration.Mean =
	mean(Instant_Accel_1$Current.Accel.Calibrated[Instant_Accel_1$Track == i])

	Acceleration.Max =
	max(Instant_Accel_1$Current.Accel.Calibrated[Instant_Accel_1$Track == i])

	#Compile all data

	Compiled = data.frame(Acceleration.Mean, Acceleration.Max)

	Acceleration = rbind(Acceleration, Compiled)

}

#End of loop


#Compile Speed and Acceleration

Speed_Accel = data.frame(Speed, Acceleration)


#Remove empty rows

Speed_Accel = Speed_Accel[-1,]


#Reset row names

rownames(Speed_Accel) = NULL


#Compile all data into one data frame

Trajr_Data = data.frame(Speed_Accel, Trajr_Data)
```

#Export data frame to working directory

```
write.csv(Trajr_Data,"Trajr_Data.csv")
```

3. Run the following code to acquire figures of the trajectory (original and smoothed) and swimming speed and acceleration over time. This code will save all of the outputs automatically into your working directory.

```
for (i in unique(Tracking_RealSpace_1$Track)) {

	Track = subset(Tracking_RealSpace_1, Track == i)

	Track_xy = Track[,c(8,9,2)]

	Track_xy$Time..sec.[1] = 0

	trj = TrajFromCoords(Track_xy, xCol = 1, yCol = 2, timeCol = 3, fps = 30,
	spatialUnits = "mm", timeUnits = "s")

	trj_smoothed = TrajSmoothSG(trj, p = 3, n = 31)

		#Make sure these match the smoothing parameters used previously

	#Plot original track

	file_name1 = paste("Track", i, ".tiff", sep="")

	tiff(file_name1, height = 20, width = 30, units='cm', compression = "lzw", res
	= 300)

	plot(trj)

	dev.off()

	#Plot smoothed track

	file_name2 = paste("Track", i, "_smoothed", ".tiff", sep="")

	tiff(file_name2, height = 20, width = 30, units='cm', compression = "lzw", res
	= 300)

	plot(trj, lwd = 1, lty = 1)

	lines(trj_smoothed, col = "red", lwd = 2)

	dev.off()

	#Calculate speed and acceleration derivatives

	derivs = TrajDerivatives(trj_smoothed)

	#Plot change-in-speed and speed

	file_name3 = paste("Track", i, "_SpeedAccel", ".tiff", sep="")
```

```
	tiff(file_name3, height = 20, width = 30, units='cm', compression = "lzw", res
	= 300)
	par(mar = c(5, 4, 4, 4) + 0.3)
	plot(derivs$acceleration ~ derivs$accelerationTimes, type = 'l', col = 'red',
		yaxt = 'n',
		xlab = 'Time (s)',
		ylab = expression(paste('Acceleration (', mm/s^2, ')')))
	axis(side = 2, col = "red")
	lines(derivs$speed ~ derivs$speedTimes, col = 'blue')
	axis(side = 4, col = "blue")
	mtext('Speed (mm/s)', side = 4, line = 3)
	abline(h = 0, col = 'lightGrey')
	title(main = paste("Track", i))
	dev.off()
}
#End of loop
```

NOTE:

Clear environment prior to setting new working directory :)

Statistical Analysis

After all videos are analyzed, compile all tracking data into one excel sheet and save as a .csv file.

Make sure there are columns with metadata (identification factors): such as treatments used, age, generation, etc.

1. Set working directory to file containing .csv file of your compiled results.

   In toolbar: Session > Set Working Directory > Choose Directory

2. Acquire descriptive statistics for each tracking metric. “Factor” represents the metadata column you want to compare.

#Upload compiled data

```
Compiled_Data = read.csv(file="Compiled_Data.csv", header=TRUE, row.names = 1)
```

#Descriptive results: mean, standard deviation, median, median absolute deviation (mad), minimum, maximum, skew, and standard error.

```
descriptive.stats = describeBy(x=Compiled_Data[,-1], group=Compiled_Data$Factor,
mat=TRUE)
```

#Export data frame to working directory.

```
write.csv(descriptive.stats, file = "Descriptive_Statistics.csv")
```

3. Use the following code to identify and remove outliers within each swimming behavior metric. “Factor” represents the metadata column you want to compare.

#Check boxplots for outliers

```
for (col_name in colnames(Compiled_Data)[-1]) {
  boxplot<-ggplot(Compiled_Data, aes(x = Factor, y = Compiled_Data[, col_name])) +
    geom_boxplot() +
    labs(title = paste("Box Plot for", col_name), x = "Factor", y = col_name)
  plot(boxplot)
}
```

###Count how many potential outliers (dots beyond the minimum and maximum whiskers) there are in each plot.

#Rosner Test for Outliers: Determine if potential outliers need to be removed

##"Metric" represents the swimming behavior metric you want to analyze for significance.

##Change k value to equal the number of potential outliers identified in the boxplots.

##Copy the following code for each swimming behavior metric with potential outliers.

```
rosnerTest(Compiled_Data$Metric, k = 1)
```

*****Results Interpretation*****

If the "Outlier" column shows a "TRUE" response then the sample needs to be removed from the analysis for the specified swimming behavior metric.

#Remove outliers from dataset by converting them to "NA"

#Create a copy of the original data frame for reference or if issues occur

```
Compiled_Data_original = Compiled_Data
```

##From rosnerTest() output use the "Obs.Num" column for "row" location in matrix

##Use metric column within data matrix for "column"

```
Compiled_Data[row, column] <- NA
```

4. Use the following code to determine what statistical test to use (parametric or nonparametric) if you are comparing more than one variable to a control.

#Shapiro-Wilk Test : Normality of Data

```
shapiro.pass = list()
for (z in colnames(Compiled_Data[,-1])){
      shapiro = Compiled_Data %>% group_by(Factor) %>% shapiro_test(z)
      shapiro.list = if (any(shapiro$p<=0.05)) { print("Nonparametric") }
           else { print("Parametric") }
      shapiro.pass[z] = shapiro.list
```

```
}
shapiro.pass = t(as.data.frame(shapiro.pass))
colnames(shapiro.pass) = c("Pass")
```

#Bartlett's Test: Homogeneity of Variances

```
bartlett = lapply(Compiled_Data[,-1], bartlett.test, Compiled_Data$Factor)
bartlett.pvalue  = data.frame(sapply(bartlett, getElement, "p.value"))
colnames(bartlett.pvalue) = c("p.value") #Shorten long column name
```

#Creates a data frame to keep track of which test to use for each metric

```
determine.test = merge(shapiro.pass, bartlett.pvalue, by = "row.names")
determine.test[,4] = ifelse((determine.test$Pass ==
"Parametric")&(determine.test$p.value > 0.05), "OneWayANOVA", "Kruskal")
colnames(determine.test) = c("metric", "shapiro", "bartlett", "Test")
                #Shortens long column name
```

5. Conduct appropriate statistical test based on the result in the “Test” column in the "determine.test" data frame. “Metric” represents the swimming behavior metric you want to analyze for significance. “Factor” represents the metadata column you want to compare. Copy the following code for each swimming behavior metric.

#Parametric Test: One-Way ANOVA

```
model_Metric = aov(Metric ~ Factor, data=Compiled_Data)
anova(model_Metric)
```

*****Results Interpretation*****

If the p-value is less than 0.05, then there are significant differences between factors. Conduct a pairwise test using the code below to determine where those significant differences occur.

If the p-value is greater than 0.05, then there are no significant differences between factors. A pairwise test should not be conducted.

#Pairwise Test for Significant One-Way ANOVA results: Tukey's Test

```
TukeyHSD(model_Metric)
```

#Non-parametric Test: Kruskal-Wallis

```
kruskal.test(Metric ~ Factor, data=Compiled_Data)
```

*****Results Interpretation*****

If the p-value is less than 0.05, then there are significant differences between factors. Conduct a pairwise test using the code below to determine where those significant differences occur.

If the p-value is greater than 0.05, then there are no significant differences between factors. A pairwise test should not be conducted.

#Pairwise Test for Significant Kruskal-Wallis results: Multiple-Comparison Test

```
kruskalmc(Metric ~ Factor, data=Compiled_Data)
```

6. Plot the results of the significance testing (boxplot visuals). "Control Factor" represents the control variable (i.e., reference group) with which other factor levels are compared.

#Parametric Data (One-Way ANOVA metrics)

```
ggdotplot(Compiled_Data, x = "Factor", y = "Metric") +
      geom_dotplot(binaxis="y", stackdir="center", dotsize = 1) +
      ylab("Metric (units)") +
      scale_y_continuous(limits = c(0, NA)) +
                  #remove the above line if y-axis does not need to start at zero
      theme(axis.title.x = element_text(margin = margin(t = 15), size = 35),
            axis.title.y = element_text(margin = margin(r = 15), size = 30),
            axis.text.x = element_text(size = 35),
            axis.text.y = element_text(size = 30)) +
      stat_summary(fun.data = "median_q1q3", geom = "errorbar", size = 1.5,
      colour = "red", width = 0.3) +
```

```
        stat_summary(fun.y = "median", geom = "crossbar", size = 1, colour =
        "red", width = 0.4) +
        geom_pwc(ref.group = "Control Factor",
              method = "t.test",
              label = "p.signif",
              hide.ns = TRUE,     #hides non-significant comparisons
              label.size = 10,    #asterik (*) size
              tip.length = 0.03, #length of bracket tip
              size = 0.5,         #bracket size
              vjust = 0.5)        #distance between bracket and asterisk (*)
```

#From image display panel select: Export > Save as PDF > "US Letter" for consistency

#Nonparametric Data (Kruskal-Wallis metrics)

```
ggdotplot(Compiled_Data, x = "Factor", y = "Metric") +
        geom_dotplot(binaxis="y", stackdir="center", dotsize = 1) +
        ylab("Metric (units)") +
        scale_y_continuous(limits = c(0, NA)) +
                    #remove the above line if y-axis does not need to start at zero
        theme(axis.title.x = element_text(margin = margin(t = 15), size = 35),
          axis.title.y = element_text(margin = margin(r = 15), size = 30),
          axis.text.x = element_text(size = 35),
          axis.text.y = element_text(size = 30)) +
        stat_summary(fun.data = "median_q1q3", geom = "errorbar", size = 1.5,
        colour = "red", width = 0.3) +
        stat_summary(fun.y = "median", geom = "crossbar", size = 1, colour =
        "red", width = 0.4) +
        geom_pwc(ref.group = "Control Factor",
              method = "wilcox.test",
              label = "p.signif",
              hide.ns = TRUE,     #hides non-significant comparisons
              label.size = 10,    #asterik (*) size
```

```
        tip.length = 0.03, #length of bracket tip
        size = 0.5,        #bracket size
        vjust = 0.5)       #distance between bracket and asterisk (*)
#From image display panel select: Export > Save as PDF > "US Letter" for consistency
```