Supporting Information to "The influence of temperature and salinity on motility of *Hematodinium perezi* dinospores released from the mudflat crabs *Helice tientsinensis*" Zhang et al. , submitted

Xutao Zhang

June 2024

Overview

This Supporting Information accompanies our submitted paper, "The influence of temperature and salinity on motility of *Hematodinium perezi* dinospores released from the mudflat crabs *Helice tientsinensis*". Below, we provide the code for video capture and edit, cell tracking in image sequences, trajectory visualization, and statistical analysis. The codes used by the two kinds of dinospores were basically the same. Here we use macrospores as an analysis example. These codes were writing using Python (version 3.8.10), ImageJ (version 1.53t), or R (version 4.3.0). Data supporting the results are provided in Appendix 1 and archived in the GitHub repository, including R and Excel files, see link: (https://github.com/ZhangXutao123/*Hematodinium*-dinospores.git).

- **Trajectory point data**: Contains the position coordinates of each trajectory in the experiment, namely "1. Microspores_TrajectoryPoints.Rdata" and "1. Macrospores_TrajectoryPoints.Rdata".
- **Density and motility data**: Derived from trajectories, namely "2.Microspores_density.Rdata", "2.Microspores_motility.Rdata", "2.Macrospores_density.Rdata", and "2.Macrospores_motility.Rdata".
- **Total movement distance**: Derived from motility data, namely "3.Microspores_distance.Rdata" and "3.Macrospores_distance.Rdata".

Video capture and edit

Video Capture

The hemocytometer (10 μL volume, 100 μm depth) was used to observe the *Hematodinium* dinospores. Following the homogenization of the samples, a volume of 10μL was distributed along the counting chamber race by capillarity to fill them completely. Images were taken using an Olympus BX53 microspore (200×), equipped with a video camera (Olympus DP73) running Cell Sens Standard 1.7 software. The resolution of the captured images was 746×578 pixels, with a relationship of 0.625 μm/pixel in both axes. Multiple fields of the central counting chamber (square printed on the slide) were recorded, amounting to no fewer than 20 fields across three replicates. Each field was recorded for a minimum of 5 seconds at a rate of 10 frames per second.

Trim the target segments.

Due to the movement of the microspore stage and slides during observation, the recorded videos contain unstable segments with motion distortion. Use FastStone Capture 9.2 video editing software (https://www.faststone.org/FSCaptureDetail.htm) to trim the original videos, saving stable field of view segments. Name these segments accurately according to the group number, sampling time, and repetitions of the samples.

Batch process videos into images.

Get the names of all the video files in the folder, then convert each video into frame images. The following code is written in Python.

Get the names of all the video files in the specified folder.

```
import shutil
import cv2
import os
# Function to get the names of all files in the specified directory
def get_file_names(file_dir):
    """
    Get the names of all files in the specified directory.
    """
    files_all = []
    for root, dirs, files in os.walk(file_dir):
        files_all.extend(files)
    return files_all
# Get the list of video files in the input directory
file_list = get_file_names('./input')
```

Convert the videos into frame images.

```
for file_name in file_list:
    print(file_name)
    # Extract the base name of the file (without the extension) for output
directory naming
    base_file_name = file_name[:-4]
    print(base_file_name)
    # Set the path to the video file
    video_path = f'./input/{file_name}'
    frame_count = 0
    # Set the frequency of frame extraction (every 1 frames)
    frame_frequency = 1
    # Define the output directory for the extracted frames
    output_dir_name = f'./output/{base_file_name}/'
    if not os.path.exists(output_dir_name):
        # Create the directory if it does not exist
        os.makedirs(output_dir_name)
    # Open the video file for frame extraction
    camera = cv2.VideoCapture(video_path)
    while True:
        frame_count += 1
        ret, frame = camera.read()
        if not ret:
            # If no frame is returned, end the extraction process
```

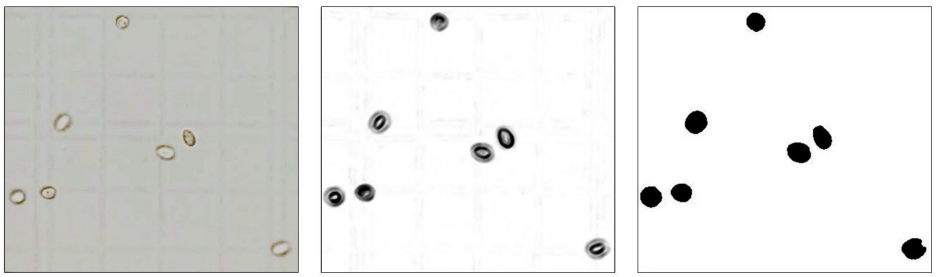
```
            print('No frame returned, ending extraction.')
            break
        if frame_count % frame_frequency == 0:
            # Save the extracted frame as a PNG file
            cv2.imwrite(f'{output_dir_name}{frame_count}.png', frame)
            print(f'{output_dir_name}{frame_count}.png is being generated.')
    print('Frame extraction completed.')
    camera.release()
    # Move the processed video file to the output directory

    shutil.move(f"./input/{file_name}", f"./output/{file_name}")
```

Cell tracking in image sequences

In ImageJ, sequentially process each group of image sequences to separate cells from the background, then use the MTrack2 plugin to detect trajectories. (shown in Fig. 2 of the manuscript). The following code is a built-in macro in ImageJ.

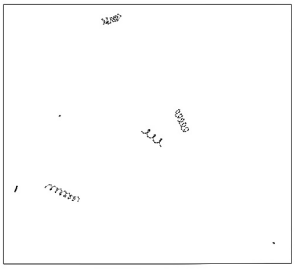
Separate cells from the background.

```
file_path="~/Video_170659_edit_1.gif";
// Import the image sequence
open(file_path);
// Convert to grayscale
run("8-bit");
// Remove characters before and after the file path for later file export
file_name=replace(file_path, "~/", "");
file_name=replace(file_name, ".gif", "");
new_file_path="~/Processed/"+file_name;
// Highlight cell edges using a filter, setting the radius to the width of the
cell edges in pixels, with macrospores set to the pixel size corresponding to 3.5
μm, and microspores set to the pixel size corresponding to 1.5 μm
run("Variance...", "radius=5 stack");
// Preliminary segmentation of cells and background using auto-thresholding
run("Auto Threshold", "method=Li white stack");
// Fill holes in cells
run("Fill Holes", "stack");
// Separate overlapping cells using watershed algorithm
run("Watershed", "stack");
save_path=new_file_path +"_black.gif";
saveAs("gif", save_path);
save_path=new_file_path +"_path.txt";
```



Use the MTrack2 plugin to detect trajectories

```
// Use MTrack2 plugin in ImageJ based on nearest neighbor algorithm to link cell position detection trajectories
run("MTrack2 ", "minimum=200 maximum=1300 maximum_=20 minimum_=0 save display show show_0 show_1 save="+save_path);
// Final specific parameters for macrospores, minSize = 3.1415×(4 μm)^2, max Size =3.1415x(10 μm)^2, max Speed= 12 μm, min TrackLength = 0 μm,
// and for microspores, minSize = 3.1415×(1 μm)^2, max Size =3.1415x(5 μm)^2, max Speed = 20 μm, min TrackLength =0 μm.
selectWindow("Paths");
save_path=new_file_path +"_path.png";
saveAs("png", save_path);
close();close();close();
```



Trajectory visualization and statistical analysis.

Trajectory visualization

Visualizing the trajectories of dinospores (shown in Fig. 3 of the manuscript). It uses the "coords_data.RData" file containing single-cell coordinate data (X, Y) to plot the trajectory.

Load required packages and set your own working directory. The following is the R code.

```
# Clear all objects in the workspace
rm(list = ls())
# Perform garbage collection to free up memory
gc()
# Load necessary libraries
library(rstudioapi)
library(ggplot2)
```

```r
# Get the current script directory and set it as the working directory
cur_dir = dirname(getSourceEditorContext()$path)
setwd(cur_dir)
getwd()
```

Write a function to translate trajectory points so that the trajectories are displayed in the first quadrant of the coordinate system.

```r
# Define a function to translate coordinates to the origin
translate_to_origin = function(coords_data) {
    # Translate the first point to the origin (0,0)
    x1 = coords_data$X[1]
    y1 = coords_data$Y[1]
    coords_data$X = (coords_data$X) - x1
    coords_data$Y = (coords_data$Y) - y1
    return(coords_data)
}
# Define a function to translate coordinates to the first quadrant
translate_to_first_quadrant = function(coords_data) {
    # Find the minimum x and y values
    x_min = min(coords_data$X)
    y_min = min(coords_data$Y)
    # Adjust coordinates to ensure they are in the first quadrant
    if ((x_min - 5) < 0) {
        coords_data$X = coords_data$X + 5 - x_min
    }
    if ((y_min - 5) < 0) {
        coords_data$Y = coords_data$Y + 5 - y_min
    }
        return(coords_data)
}
```

Write a function to calculate trajectory speeds and assign colors to the trajectories based on their speeds.

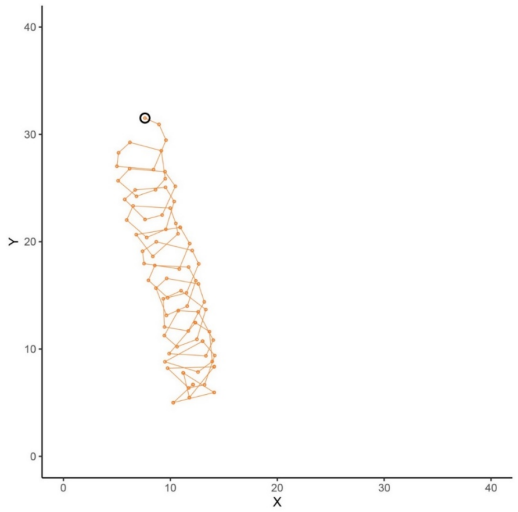
```r
# Define a function to calculate the average speed
calculate_average_speed <- function(coords_data) {
    # Initialize total distance
    total_distance <- 0
    # Iterate through each pair of consecutive points to calculate the distance
    for (i in 2:nrow(coords_data)) {
        distance <- sqrt((coords_data$X[i] - coords_data$X[i-1])^2 +
                         (coords_data$Y[i] - coords_data$Y[i-1])^2)
        total_distance <- total_distance + distance
    }
    # Calculate the average speed
```

```
    average_speed = total_distance / (nrow(coords_data) * 0.1)
    return(average_speed)
}
# Define a function to plot the cell trajectory
plot_cell_trajectory = function(coords_data, speed_color) {
    # Translate coordinates to the origin and then to the first quadrant
    coords_data = translate_to_origin(coords_data)
    coords_data = translate_to_first_quadrant(coords_data)
        # Get the last point (end point of the trajectory)
    first_point = tail(coords_data, n = 1)
        # Create the plot
    p = ggplot(data = coords_data, mapping = aes(x = X, y = Y)) +
        geom_path(data = coords_data, mapping = aes(x = X, y = Y),
                  colour = speed_color,
                  size = 0.2) +
        geom_point(data = coords_data, mapping = aes(x = X, y = Y),
                   shape = 21,
                   colour = speed_color,
                   fill = NA,
                   stroke = 0.5,
                   size = 0.7) +
        geom_point(data = first_point, mapping = aes(x = X, y = Y),
                   shape = 21,
                   colour = "black",
                   fill = NA,
                   stroke = 1,
                   size = 3) +
        xlim(0, 40) +
        ylim(0, 40) +
        theme_classic()
    return(p)
}
```

Plot the trajectory.

```
# Load required data, coords_data.RData contain a dataframe with X and Y
coordinates
load("../coords_data.RData")
# Define a color palette
color = c('#30123B', '#3A2D7A', '#4146AD', '#455FD4', '#4677EF', '#438EFD',
'#38A5FA', '#28BCEA', '#1BD0D3', '#18E0BC', '#26EDA4', '#42F687', '#65FC68',
'#88FE4D', '#A5FC3B', '#BDF434', '#D3E734', '#E6D738', '#F5C43A', '#FCB035',
'#FE972B', '#FA7B1F', '#F26014', '#E6490B', '#D73606', '#C42502')
# Calculate the average speed and determine the color based on speed
average_speed = ceiling(calculate_average_speed(coords_data))
```

```
speed_color = color[average_speed]
# Plot the trajectory
p = plot_cell_trajectory(coords_data, speed_color)
# Save the plot as a PDF file
ggsave(paste('./', 'plot', '.pdf', sep=''), p, width = 6, height = 6)
```



Line chart

The following is a dual line chart of cell density and motility (shown in Fig. 4 of the manuscript). Load required packages and set your own working directory. Load the required data and convert the units to meters. The following is the R code.

```
# Clear all objects in the workspace
rm(list = ls())
# Perform garbage collection to free up memory
gc()
# Load necessary libraries
library(ggplot2) # Plotting tool
library(rstudioapi)
# Get the current script directory and set it as the working directory
cur_dir = dirname(getSourceEditorContext()$path)
setwd(cur_dir)
getwd()
# Load data
# The dataset `all_data` contains columns: Group, Time, temperature,
salinity, data_mot, data_mot_error, data_num, data_num_error
load("./all_data.Rdata")
# Convert data to meters
all_data$data_mot = all_data$data_mot * 10^(-6)
all_data$data_mot_error = all_data$data_mot_error * 10^(-6)
```

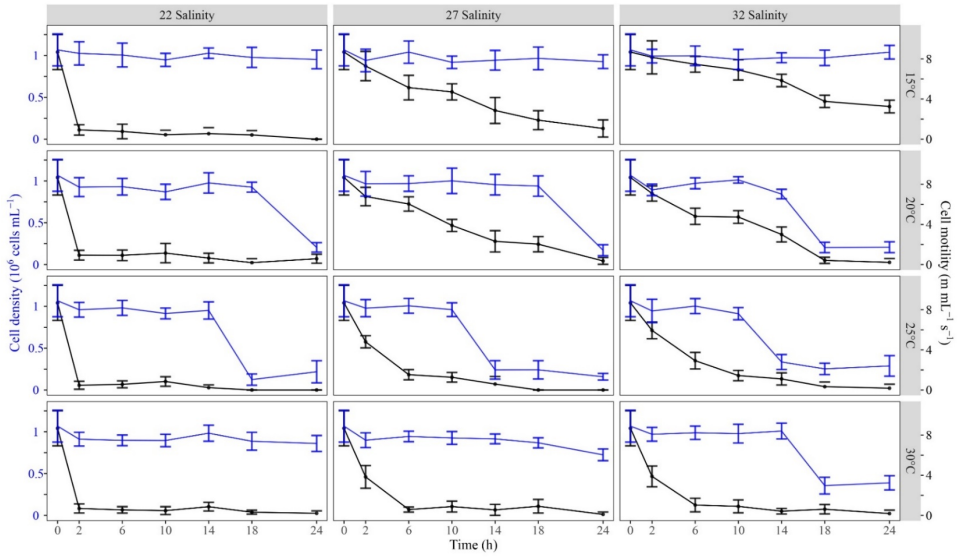
Calculate the total movement distance for each group and assign colors based on the total movement distance of each group.

```
# Compute the integral area for each group, i.e., the total movement
distance for each group.
Integral_color = data.frame()
for (group_i in 1:12) {
  data_i = subset(all_data, all_data$Group == group_i)
  # Create a data frame with x and y coordinates
  df <- data.frame(x = c(0, data_i$Time, 24), y = c(0, data_i$data_mot,
0))
  # Ensure the polygon is closed
  df <- rbind(df, df[1, ])
  df$x = df$x * 3600
  # Create a polygon object based on coordinates
  poly <- Polygon(df)
  # Calculate the area between the polygon and the x-axis
  area <- abs(poly@area)
  print(group_i)
  print(area)
  row_data = data.frame(Group = group_i, area = area)
  Integral_color = rbind(Integral_color, row_data)
}
# Map the area data to gradient colors
Integral_color$area_color <- colorRampPalette(c("firebrick3", "white",
"navy"))(52)[floor(Integral_color$area * 0.0001) + 1]
sorted_Integral_color <- Integral_color[order(Integral_color$area), ]
```

Scale down the y-axis for cell motility and create a dual y-axis line plot.

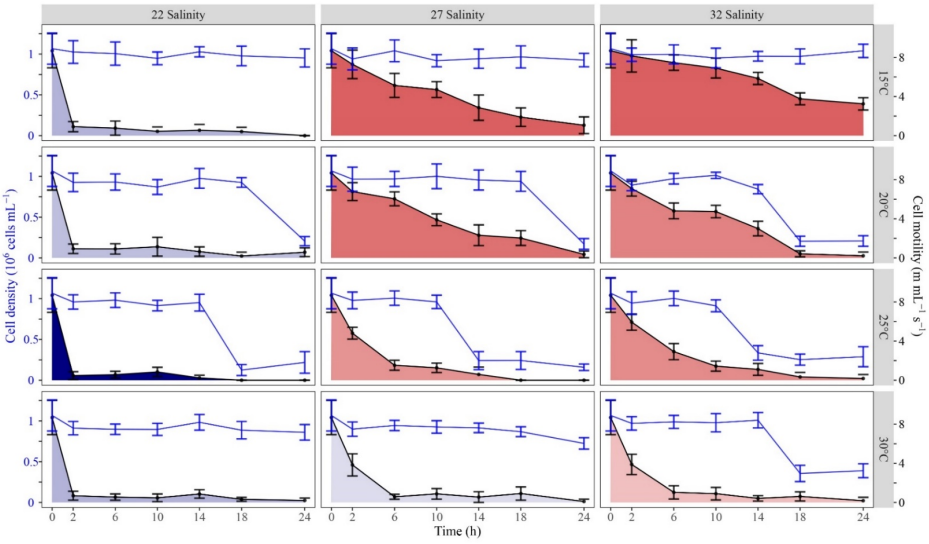
```
# Prepare data for plotting
plot_data = all_data
plot_data$data_mot = plot_data$data_mot * 120000
plot_data$data_mot_error = plot_data$data_mot_error * 120000
plot_data <- merge(plot_data, Integral_color, by = "Group", all.x = TRUE)
plot_data$area = round(plot_data$area * 0.01)
plot_data$area = factor(plot_data$area, levels =
sort(unique(plot_data$area)))
# Plot cell density and cell motility on dual axes
p <- ggplot(plot_data, aes(x = Time)) +
  geom_line(aes(y = data_mot), color = "black") +  # Plot line for
motility data
  geom_point(aes(y = data_mot), color = "black", size = 1, alpha = 0.8) +
# Plot points for motility data
  geom_errorbar(aes(ymin = data_mot - data_mot_error, ymax = data_mot +
data_mot_error), color = "black", size = 0.75, width = 1, alpha = 0.8) +
# Error bars for motility data
```

```
  geom_line(aes(y = data_num), color = 'blue', stat = "identity", alpha = 0.8) +  # Plot line for cell density
  geom_errorbar(aes(ymin = data_num - data_num_error, ymax = data_num + data_num_error), color = 'blue', size = 0.75, width = 1, alpha = 0.8) + # Error bars for cell density
  scale_y_continuous(
    limits = c(0, 1.3 * 10^6),
    breaks = c(0, 0.25 * 10^6, 0.5 * 10^6, 0.75 * 10^6, 1 * 10^6, 1.25 * 10^6),
    labels = c(0, "", 0.5, "", 1, ""),
    sec.axis = sec_axis(
      ~ . / 120000,
      name = bquote("Cell motility (" * m * ' ' * mL^-1 * ' ' * s^-1 * ')'),
      breaks = c(0, 2, 4, 6, 8),
      labels = c(0, '', 4, '', 8)
    )
  ) +
  theme(axis.text.y = element_blank(), axis.text.y.left = element_text(color = "blue")) +
  theme(axis.text.y = element_blank(), axis.text.y.right = element_text(color = "black")) +
  facet_grid(vars(temperature), vars(salinity)) +
  scale_x_continuous(expand = c(0, 0.5), breaks = c(0, 2, 6, 10, 14, 18, 24)) +
  theme(legend.position = "none") +
  labs(x = 'Time (h)', y = bquote("Cell density (" * 10^6 * ' ' * cells * ' ' * mL^-1 * ')')) +
  theme(axis.title.y.left = element_text(color = "blue"), axis.title.y.right = element_text(color = "black")) +
  theme(
    panel.grid = element_blank(),
    panel.background = element_rect(color = 'black', fill = 'transparent'),
    strip.text = element_text(size = 12)
  ) +
  theme(
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 13),
    legend.title = element_blank(),
    legend.text = element_text(size = 11)
  ) +
  theme(text = element_text(size = 12, family = "serif"))
```



The following part of the code adds a shaded area under the motility curve.

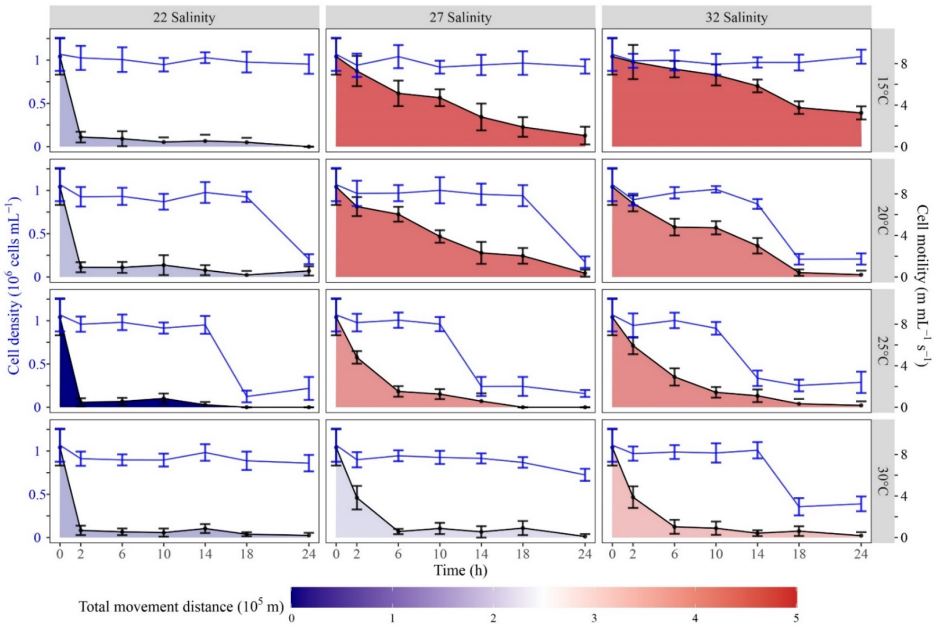
```
p = p + geom_area(aes(y = data_mot, fill = area), alpha = 1) +  # Plot shaded
area
  scale_fill_manual(values = rev(sorted_Integral_color$area_color))  # Manual fill
for area colors
```



The following code adds a legend to the plot.

```
# Print the plot
print(p + theme(legend.position = 'none'), vp = viewport(x = 0.48, y = 0.53,
width = 0.87, height = 0.93))
# Function to extract legend from a ggplot
g_legend <- function(gg_plot) {
  tmp <- ggplot_gtable(ggplot_build(gg_plot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == 'guide-box')
  leg <- tmp$grobs[[leg]]
  leg
```

```
}
# Create legend for the plot
p_legend <- ggplot(data.frame(expand.grid(X = 0:10, Y = 1:10)), aes(X, Y, fill =
runif(110))) +
  geom_tile(color = "white", size = 0.5) +
  scale_fill_gradientn(colors = c("navy", "white", "firebrick3"),
                       name = bquote("Total movement distance (" * 10^5 * " m)"),
                       limits = c(0, 50),
                       breaks = c(0, 10, 20, 30, 40, 50),
                       labels = c("0", "1", "2", "3", "4", "5")) +
  theme(text = element_text(size = 13, family = "serif")) +
  theme(panel.grid = element_blank(), panel.background = element_rect(color =
'black', fill = 'transparent')) +
  theme(legend.key.width = unit(3, "cm")) +
  theme(legend.position = "bottom", legend.direction = "horizontal")
# Extract and draw the legend on the plot
phylum_legend <- g_legend(p_legend)
phylum_legend$vp$x <- unit(0.452, 'npc')  # Specify x position of the legend
phylum_legend$vp$y <- unit(0.032, 'npc')  # Specify y position of the legend
grid.draw(phylum_legend)
```



Correlation analysis of total movement distance, temperature, and salinity

The total movement distance for each group has been calculated. We will perform a correlation analysis between the total movement distance, temperature, and salinity. The following is the R code.

```
# Clear the environment and run garbage collection
rm(list = ls())
gc()
```

```r
# Load necessary libraries
library(rstudioapi)
# Load the Hmisc package for correlation analysis
library(Hmisc)
# Set the working directory to the current script's directory
cur_dir = dirname(getSourceEditorContext()$path)
setwd(cur_dir)
getwd()
# Load the data, which includes three columns: distance, temperature, and
salinity
load("./distance_data.Rdata")
# Calculate Pearson correlation coefficients
res2 <- rcorr(as.matrix(distance_data), type = c("pearson"))
# Print the correlation coefficients and p-values
res2$r
res2$P
```

```
> # Print the correlation coefficients and p-values
> res2$r
            Distance temperature salinity
Distance     1.000000   -0.606593  0.59594
temperature -0.606593    1.000000  0.00000
salinity     0.595940    0.000000  1.00000
> res2$P
                Distance temperature     salinity
Distance              NA 8.79101e-05 0.0001255754
temperature 0.0000879101          NA 1.0000000000
salinity    0.0001255754 1.00000e+00           NA
```

Fitted plot

The total movement distance for each group has been calculated. Use polynomial fitting to fit the total movement distance under different treatments (shown in Fig. 5 of the manuscript).
Load required packages and set your own working directory. Load the total movement distance data for each group. The following is the R code.

```r
# Clear the workspace and run garbage collection to free memory
rm(list = ls())
gc()
# Load required libraries
library(ggplot2)  # Core plotting library
library(rstudioapi)
cur_dir = dirname(getSourceEditorContext()$path)
setwd(cur_dir)
getwd()
# Load the data. The data includes the number of groups, temperature, salinity,
and the corresponding total movement distance.
load("./plot_data")
```

Fit the total movement distance within the experimental temperature and salinity range.

```
# Fit a polynomial regression model using the lm() function
attach(plot_data)
fit = lm(distance ~ temperature + salinity + I(temperature^2) + I(salinity^2) +
temperature:salinity, data = plot_data)
summary(fit)
# Extract coefficients from the model
fit$coefficients
coefficients = as.numeric(fit$coefficients)
# Define a function to predict total movement distance based on the polynomial
model
f <- function(x, y) {
  r = coefficients[1] + coefficients[2] * x + coefficients[3] * y +
coefficients[4] * x * x + coefficients[5] * y * y + coefficients[6] * x * y
  return(r)
}
```

Plot the fitted results.

```
# Create a grid for plotting
temperature_seq = seq(15, 30, length.out = 150)
salinity_seq = seq(22, 32, length.out = 150)
grid = expand.grid(temperature = temperature_seq, salinity = salinity_seq)
# Predict total movement distance for each combination of temperature and
salinity
grid$distance_pred = apply(grid, 1, function(x) f(x[1], x[2]))
# Plot the data using ggplot2
p = ggplot(grid, aes(x = salinity, y = temperature, fill = distance_pred)) +
  geom_raster() +
  scale_fill_gradientn(colours = c("navy", "white", "firebrick3"), na.value =
"grey50",
                       name = bquote("Total movement distance (" * 10^5 * '·'*m
*')') ,
                       limits = c(0, 50),
                       breaks = c(0, 10, 20, 30, 40, 50),
                       labels = c("0", "10", "20", "30", "40", "50")) +
  labs(x = "Salinity", y = "Temperature (°C)", fill = "Growth") +
  scale_x_continuous(position = "top",
                     breaks = c(22, 24, 26, 28, 30, 32),
                     labels = c(22, 24, 26, 28, 30, 32)) +
  scale_y_continuous(position = "right", trans = "reverse",
                     breaks = c(15, 20, 25, 30),
                     labels = c(15, 20, 25, 30)) +
  theme(panel.border = element_blank(),
        panel.grid.major = element_blank(),
```

```
        panel.grid.minor = element_blank(),
        axis.line = element_line()) +
  theme(panel.grid = element_blank(), panel.background = element_rect(color =
'black', fill = 'transparent')) +
  theme(legend.key.height = unit(2, "cm")) +
  theme(panel.grid = element_blank(),
        panel.background = element_rect(color = 'black', fill = 'transparent'),
        strip.text = element_text(size = 12)) +
  theme(axis.text = element_text(size = 12),
        axis.title = element_text(size = 13),
        legend.text = element_text(size = 11)) +
  theme(text = element_text(size = 12, family = "serif"))
# Print the plot
print(p)
```