Testing Zebrafish swimming randomness using Markov Chain entity tracking

Can Erozer, Jackson Fisk

Initial Ideas & Goals:

During the semester we learned techniques about the next position estimation of the objects, i.e. kalman and alpha-beta filter. And our primary goal is to experiment with different techniques to predict the next movement of objects. As students for the Stochastic Processes course we want to try if Markov Chains would be a good choice for achieving this.

Goal 1: Experiment if Markov Chains would be a good tool for object position estimation rather than the tools we learned in class.

Due to the definition of Stochastic processes, it is a collection of states indexed by time. So, the object that we are going to estimate its next position should be an object whose position is, mostly, random. So, our domain for picking an object should be among insects, animals, bacterias, or particles. This is because there is much research that states the movement of animals, bacteria and particles follows a random walk (though not only the "classical" random walk).[1][2][3]

However, in our research, we learned that the randomness in the movements of animals and bacterias can be affected by external factors such as geography, predators, food sources, and other surrounding animals. We thought this is very interesting and we decided to experiment this too:

Goal 2: Observing how the movements of animals deviates from randomness as external factors get involved.

Dataset:

Because our time was limited, we tried to find a dataset on the web. And one of the best datasets we found is the one that tracks the positions of Zebrafish over time. This was the most suitable data we found because the Zebrafish were kept in a void tank (no obstacles, no food source, e.g. nothing that could prevent their movements) and it gives the 2D view of the fish from the top (not a perfect 2D projection but still a decent projection). Two types of data provided: the positions of the fish in a tank with two

fishes inside and in another tank with five fish inside. This variability helps us for our second goal: We can observe how randomness in fish's movements get affected as other fish exist in the environment (an external factor).

The dataset offers the video of the fish, the 2D position of them in each frame, and the position of the bounding boxes.

Model Architectures:

We had two choices when it comes to deciding the type of state spaces: continuous or discrete. In the domain of stochastic processes, for continuous state space, there are existing models like brownian motion. But these techniques are beyond our understanding and spending too much time on understanding it would be out of the scope of this project.

On the other hand, for discrete state space, we have easier options like discrete-time and continuous-time Markov Chains. To go with continuous-time markov chains, we needed further information about the distribution of Zebrafish movements. Even though Gaussian distribution could be a good starting point and reasonable choice of distribution, we need to go over the data to estimate its parameters. We didn't like this approach because we have limited data and a pre-iteration would be like cheating (We don't have enough data for the training set. And visiting the test-set could create a bias).

Goal 3: We didn't want to create a case-specific model. Instead, we aimed to create a model that could be used for any object whose motion tends to be random.

Therefore, we decided to go with discrete-time Markov Chains. This made sense because the time is discrete in our experiment.

The idea behind our definition of a state is that each small square (all identical areas) in the frame can be represented as a state. Here is the visualization of the states: (IMAGE OF GRID)

In order to find the number of states, we used the following formula:

$$N = \frac{area}{x^2}$$

where N is the number of states, video area is the total area of the video (width * height), and x is the pixel length of the sides of each state (each state represents a square area in the frame). Our video feed was 664x676, so the total area is 448864 pixels (The decision of the width and height choice is discussed in the "Choice of Parameters" section). We need a small value of x such that x^2 divides the area of the area into a whole number. So we found the possible small x values: [1, 2, 4, 13, 26, 52]. The options 13, 26 and 52 are too big. So we had to decide between 1, 2 and 4.

We treated this variable as a hyperparameter. And overall the best results are yielded as we picked x=1. We will discuss this choice in the "Choice of Parameters" section. Regardless of the RMSE scores we get depending on the choice of this parameter, different choices of x affected the runtime/ efficiency of our model. As x gets smaller, n, the number of states, gets bigger. And as n gets bigger the number of iterations to fill the transition probability matrix increases. To give an example, when we picked x=1, the runtime for getting the results was approximately 20 seconds. But, when x=4, the runtime decreased to less than 1 seconds.

As we decided upon the values of x and n, we needed to fill the elements of the transition probability matrix. We created our first implementation using a uniform distribution, with the transition probabilities being defined as 1 / # of possible states. For instance:

S1 (p: 1/9)	S2 (p: 1/9)	S3 (p: 1/9)
S4 (p: 1/9)	S5 (p: 1/9) *fish*	S6 (p: 1/9)

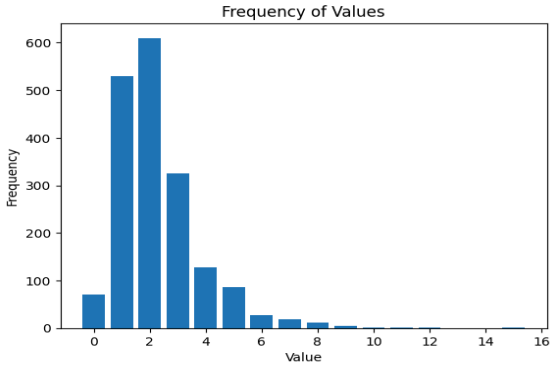
S7 (p: 1/9)	S8 (p: 1/9)	S9 (p: 1/9)

For example, imagine our image is the top down view of a fish tank, and it is divided into a 3x3 grid of states. If the fish is at state s5 at time t, then, we assume that fish can only go to the adjacent states or stay in its state. And assuming the fish's movement is completely random it can move to or stay in states with equal chances. Another example is, when the fish at the corner state:

S1 (p: 1/4) *fish*	S2 (p: 1/4)	S3 (p: 0)
S4 (p: 1/4)	S5 (p: 1/4)	S6 (p: 0)
S7 (p: 0)	S8 (p: 0)	S9 (p: 0)

If the fish is in a corner cell, it only has access to three other cells immediately and then itself, so the probabilities are all ¼.

We wanted to validate our assumption that the fish can only move its adjacent states, and not beyond, between frame t and t+1. This assumption is natural since the time between frames of the video is very small. But it didn't turn out as we thought initially. To show, look at the distribution of fish's displacement between respective frames:



This bar chart shows the frequency of displacement of a fish between frames. The values are obtained from one video, by tracking one of the fish. So the most likely displacement of the fish is 2 pixels in this case. And depending on the value of x, the side length of the square that represents a state, the fish can go beyond its immediate neighbor states. This is not a desirable thing according to our model's architecture. This is because we create the transition probability matrix with the assumption that the fish can only move the adjacent states. In this sense, if we pick x=1, meaning each pixel is a state, the probability that the fish can go beyond the neighbor states is: 70%. Calculation:

$$Pr(going\ beyond\ neighbors) = (1 - (p_0 + p_1)) = 1 - (0.014 + 0.29) \approx 0.70$$

po is the probability of displacement=0 and p1 is the probability of displacement=1. If x=1 and displacement>1, then the fish goes beyond. This probability is too much. So regardless of the low RMSE score that x=1 results, picking x=1 would not be a consistent choice with our assumptions and model architecture.

Therefore, we needed to pick a bigger value for x. The other possible values for x are 2 and 4. The higher the value of x, the lower chance of fish could go beyond the neighbor states. So we picked x=4.

Therefore, with x=4, width=676, and height=664, N, the number of states becomes 28054.

However, the distribution shown above is an intuitive proof that the Silverfish doesn't actually move in a uniformly random way.

Improvements on the Model:

In order to change the probability distribution of the position change of Silverfish, without looking at the data, we came up with an idea. An idea we had was to take into account the direction vector of how the fish were moving in the last frame, and have the transitions in the direction of that vector have favored some possible states. Using the angle and slope of the vector, we were able to determine whether the fish was moving up, left, down or right, and then we scaled the probabilities in that direction by the hyperparameter $\boldsymbol{\alpha}$, then normalized the probabilities.

For instance, say the center of the fish went from coordinates (50,50) to (60,75).

$$slope = \frac{\triangle y}{\triangle x} = \frac{75 - 50}{60 - 50} = 2.5, \theta = \arctan(2.5) = 68.19\ degrees$$

After the if conditions:
if 0<angle<=45 return “right”, elif 45<angle<135 return “up”,
elif 135<angle<225 return “left”, elif 225<angle<315 return “down”, else return “right”

(see find_direction() in fish_estimator_with_alpha.py)

After calculations, we understand that the fish went upwards in its previous movement. Because any of the directions covers the ¼ of a circle, we need to take 6 states into consideration which are s1, s2, s3, s4, s5, and s6. After favoring these probabilities by **α=2**, we have:

S1 (p: 2/9)	S2 (p: 2/9)	S3 (p: 2/9)
S4 (p: 2/9)	S5 (p: 2/9) *fish*	S6 (p: 2/9)
S7 (p: 1/9)	S8 (p: 1/9)	S9 (p: 1/9)

And the after normalization of probabilities:

S1 (p: 2/15)	S2 (p: 2/15)	S3 (p: 2/15)
S4 (p: 2/15)	S5 (p: 2/15) *fish*	S6 (p: 2/15)

S7 (p: 1/15)	S8 (p: 1/15)	S9 (p: 1/15)

One note is that, we don't change the elements of the transition probability matrix as due to the markov property. Markov Property says that the next state of a process only depends on its current state. If we were to change the transition probability matrix each time, then we would harm the Markov Property. Therefore, instead we just change **weights**:

pred_state=random.choices(possible_states, **weights=new_probabilities**,k=1)[0]

Choice of Parameters:

Width and Height:

Videos are resizable. And we cropped the video in order to get the desirable video area. By desirable video area, we mean the number that can be divisible to x^2, x is a whole number, and gives a good range of different values of x. Taking into consideration the positions of the fish, we decided that (h:664,w: 676) is a good resizing. It gave us the flexibility to take x as 1,2, and 4.

x:

In the model architecture part, we discussed why we have a limited range of choices for the x value. In this section, we want to discuss why x=1 yields the lowest RMSE scores even though it makes us deviate from the assumption that the fish can only move the neighboring states. This is because when we set x=1, we make a guess that is really close to the actual position of the fish. And the distribution shown above shows that the next position of the cannot be that far away from the original position. And, by nature, it results in us to have small values of RMSE score. Although this seems like a desirable result, it is actually not because it is not consistent with our model assumptions and it is not efficient.

Alpha:

We treated the alpha parameter as a hyperparameter. After trying different alpha values and getting different RMSE values, we found out that alpha=10 gives the lowest RMSE score. So we used alpha=10 to run our model.

We summed up all the RMSE values for each prediction of the fish. And we got the following:

```
{2: 23055.77683455478,
 3: 22619.224323324677,
 4: 22363.989143136965,
 5: 22218.68988431917,
 6: 22292.306218279013,
 7: 22097.84759471618,
 8: 22026.686578612607,
 9: 22190.64339492595,
 10: 21941.80794316974}
```

Since alpha=10 gives the lowest RMSE value, we picked the value 10 for the alpha value accordingly.

Results:

As we stated above we make the predictions in terms of states. But in the dataset the positions of the fish are given as (x,y) coordinates. So, in order to map the states into (x,y) coordinates, we found a formula that takes the state and returns the center (x,y) point of the state. (see give_centroids() in construct_transition_probability_matrix.py)

In order to compare the performance of our model, we got results from the kalman filter model as well.

We used two videos named z3 and z4 in the dataset. In z3 there are two fish and in z4 there are five fish.

For z3, the mean RMSE score is (average of RMSE scores for two fish) 584.
For z4, the mean RMSE score is (average of RMSE scores for five fish) 555.

See results/z3_kalman and results/z4_kalman

...