Telemetry Filtering Standard Operating Procedure

R interface

Built for out-migrating Chinook smolts fitted with radio telemetry transmitters

This filtering process was built based on out-migrating Chinook smolts. These fish leave natal streams and have very little upstream movement capabilities. Therefore, this filtering routine should work well for other fish expressing similar migration tactics (e.g., any juvenile anadromous emigrant salmonid). This will have to be modified if you are dealing with a fish that can move both up and downstream.

These tags are operating at a 6 pulse per minute pulse rate. Each pulse includes a power pulse and two code pulses. All data is stored in the receiver as a HEX file and then converted to a text file using Tracker software.

1. I would like a function to read in the text files from Sharepoint folders holding each site’s data just like from Dropbox. The columns should be Date Time, Site, Valid reading (Good (1) or bad (0)), channel, tag code, then signal strength.
   1. I think creating two additional separate columns for Date and Time might be helpful while keeping the Date Time column preserved as well.
      1. **Here is where I would like the first dataframe = (raw\_text\_df)**
2. From the raw text df, first I would remove any invalid detections if present Valid = 0.
3. Next, I would like to choose what integer to round to the nearest 5, 10, or whatever we might want to make this flexible. As the 2019\_2020 data I would want rounded to the nearest 10 while 2018\_19 would need to be rounding to the nearest 5. This part might be a little tricky as I would like to run all data from all 4 years with the same function if possible.
   1. I also think this is a good place to subset the data. Here you would have to make sure you have the tag releases csv read into the console so you could reference that data frame to filter out the fish, test tags, timer tags, noise, and sentinel tags.
      1. So, if tag\_round = 10 then
         1. 5,6,7,8,9 = round up
         2. 1,2,3,4 = round down
      2. If tag\_round = 5 then
         1. 1,2 = round down to 0
         2. 3,4 = round up to 5
         3. 6,7 = round down to 5
         4. 8,9 = round up to 0
      3. **Fish df:** round to the nearest 10, arranged by tag #, then Date\_Time.
         1. There should be a fish release csv read into the local environment that contains designations of fish or test tags.
      4. **Test df:**round to the nearest 10, arranged by tag #, then Date\_Time.
         1. There should be a fish release csv read into the local environment that contains designations of fish or test tags.
      5. **Timer df:** round to the nearest 5 or maybe from 571-579, arranged by tag #, then Date\_Time.
         1. Timer tags would be ending in 575 typically but we would need to round.
      6. **Noise df** round to the nearest 5 or maybe from 991-999, arranged by tag #, then Date\_Time.
         1. Noise readings would be ending in 995 typically but we would need to round.
      7. **Sentinel df** round to the nearest 5, arranged by tag #, then Date\_Time.
         1. Sentinel tags should be ending in 525 typically but I would need to round.
4. As you can see in this example straight from the raw text file I would like all tags just as the ones below 9180 is wandering from 9184 to 9175 I would want all of those rounded to the nearest 10, or to 9180.
5. I then filter out any fish data that started before the activation of the tags and after the expected conclusion of the study.
6. I would also like to compress detections from the last detection of that tag (e.g., 1240) to a variable time interval maybe standardized at 2 minutes of the last detection. It would be very helpful if you could make the function flexible to where we could change the time to compress.
7. **The would be the next checkpoint for** **dataframes**
   1. Compressed Fish df
   2. Compressed Test tag df
   3. Compressed Timer df
   4. Compressed Noise df
   5. Compressed Sentinel df
8. All tags within 2 minutes of the same tag number should be compressed into one record. So for example the end product from the sample below would have

**Tag Site Start End n**

9190 BC1 23/10/19 21:20:12 23/10/19 21:20:22 3

9190 BC1 23/10/19 21:23:32 23/10/19 21:23:32 1

1. BC1 23/10/19 22:13:28 23/10/19 22:19:28 21

23/10/19 21:20:12 BC1 1 9 190 -105

23/10/19 21:20:22 BC1 1 9 190 -114

23/10/19 21:20:32 BC1 1 9 190 -114

23/10/19 21:23:32 BC1 1 9 190 -113

23/10/19 22:13:28 BC1 1 9 180 -076

23/10/19 22:13:48 BC1 1 9 177 -114

23/10/19 22:14:38 BC1 1 9 176 -115

23/10/19 22:14:48 BC1 1 9 184 -113

23/10/19 22:14:58 BC1 1 9 175 -117

23/10/19 22:15:08 BC1 1 9 174 -111

23/10/19 22:15:18 BC1 1 9 175 -116

23/10/19 22:15:28 BC1 1 9 180 -115

23/10/19 22:15:48 BC1 1 9 181 -113

23/10/19 22:15:58 BC1 1 9 180 -118

23/10/19 22:16:18 BC1 1 9 180 -116

23/10/19 22:16:38 BC1 1 9 180 -112

23/10/19 22:16:48 BC1 1 9 180 -116

23/10/19 22:16:58 BC1 1 9 180 -119

23/10/19 22:17:08 BC1 1 9 180 -112

23/10/19 22:17:38 BC1 1 9 177 -118

23/10/19 22:18:08 BC1 1 9 180 -115

23/10/19 22:18:28 BC1 1 9 180 -118

23/10/19 22:18:38 BC1 1 9 181 -111

23/10/19 22:18:48 BC1 1 9 181 -114

23/10/19 22:19:18 BC1 1 9 180 -117

23/10/19 22:19:28 BC1 1 9 180 -116

I know you are aware but just to highlight we will be pulling multiple text files so keeping the locations separate will be important, this example is BC1 (Below Confluence) should all be in the same folder so you could name each site by using the folder name, DD1 Dahle’s Diversion etc….

1. I would then have another filter to remove any detection that is less than whatever number you decide. For this study we went with 3 detections and with the 2 minute compression that we did above this results in any detection that is less than 3 detections per 2 minutes being designated as “ghost tags or noise” and removed. This number of detections was based off some sampling of what a normal fish tag would look like passing our arrays if it was just floating with the current. I would like this function to be flexible though for multiple applications.
   1. Also I would like this filter to have an argument to filter out detections that occurred before the tag activation. This would just be a filter that allows you to designate a time and date.
      1. \*Note make sure the release times are accurate and you are within the same time zone or you chance filtering out good data.
2. This is a good intermediate point to finalize observations before removing upstream movement and building capture histories. Therefore, I would like an additional dataframe containing just the data I designate as perceived ‘valid’ detections with a tag\_id, site, start, end, n. Observations should be sorted by tag\_id and start time.
   1. Dataframe could be called **fish\_obs\_df**
3. Now I would create dataframes for building capture histories and summaries.
   1. New Dataframe **fish\_obs\_sites**
   2. Group detections that are at the same site within the same week.
   3. merge capture/release data to have release date in this dataframe
   4. Define order of sites
   5. Filter out upstream movements from this dataframe but create a separate dataframe called **upstream\_movement\_df**. Again this is specific for my study species.
      1. I am pretty sure you can just slightly modify the existing code that Kevin wrote.
   6. **Number of detections of each tag at each site per week** (this will be used as a diagnostic tool to make sure we have good data with no tag wondering).
      1. Determine the number of detections that each fish was detected at each site each week.
      2. Create a wide formatted dataframe with this information.
         1. Dataframe would have week, tag id, then each site moving from left to right in the same upstream to downstream order.
      3. Create dataframe called **detections\_per\_week**
   7. **Capture Histories** (we have 2 different release sites, so I need to be able to define different orders and sites for each release area)
      1. Use fish\_obs\_sites df
      2. Define order of sites
      3. Build and export final capture histories both in a long form with tag\_id, site\_id, and time and a wide format that can be used in a CJS or similar model. The user could then examine those capture histories for problematic observations and flag any as ‘incorrect’. Something similar to PITcleanr. Look into abnormalities or missing data moving from most upstream site to most downstream site.
   8. **Unique fish at each site per week**
      1. Determine the number of unique fish that were detected at each site each week.
      2. Create a long formatted dataframe with this information.
      3. Create a faceted bar graph in ggplot with unique fish passing each site per week.