Grupo de xeotecnoloxía aplicada

DOCUMENTACIÓN DE CÓDIGO

RailwayHeurisctSegmentation

PROYECTO:	Safeway 2020
OBJETIVO:	Segmentación heurística ferrocarril
ARTÍCULO:	
LENGUAJE:	Matlab
DESARROLLADOR:	Daniel Lamas Novoa
FECHA:	09/04/2021

Universidade Vigo

CONTENIDO

RESUMEN

El código descrito es este documento tiene como objetivo el de segmentar los elementos presentes en entornos de ferrocarril a partir de nubes de puntos en formato *.las* y de la trayectoria del sensor en formato *ASCII*. Como resultado, se obtienen los índices de los puntos pertenecientes a cada elemento, indicando de qué tipo de elemento se trata.

El proceso consta de un seccionado, un voxelizado y una segmentación heurística. En el proceso de seccionado, las nubes se organizan en fragmentos de una longitud determinada y limitando la curvatura máxima de la vía del tren en ella. Además, los puntos lejanos a la trayectoria no se incluyen en ninguna sección, ya que no aportan información de la infraestructura.

Tras voxelizar las secciones, se lleva a cabo el proceso de segmentación heurística. En este proceso se extrae secuencialmente cada tipo de elementos de cada sección.

El resultado se guarda en un archivo .mat. Este formato consta de celdas en las que en cada una se encuentra cada tipo de elemento. Dentro de estas se encuentran agrupados los índices en la nube de puntos originales de los puntos que conforman cada elemento.

Este archivo también tiene otra variable sobre el status del proceso de segmentación.

EJECUCIÓN

Para su ejecución se debe de tener el código principal *RailwayHeuristicSegmentation* y las carpetas *Classes*, *Preprocessing* y *Railway* con sus respectivas subcarpetas y códigos.

En el código principal *RailwayHeuristicSegmentation* se deben indicar las rutas de los archivos de trayectoria y de las nubes, y la ruta para guardar las salidas. También se debe indicar la ruta de las nubes modelo de los archivos. Las nubes modelo empleadas están en la carpeta *Railway\Models.*

El código se ejecuta en un bucle paralelo. Puede ser necesario configurar cómo se desea hacer el bucle paralelo.

La salida es la nube .las de entrada en la que se modifican los siguientes campos.

- Record.classification: asigna un número a cada punto en función del tipo de elemento al que pertenezca
  - Rail = 1
  - Catenaria = 2
  - Contacto = 3
  - Dropper = 4
  - Otros cables = 5
  - Mástiles = 6
  - Señales = 7
  - Semáforors = 8
  - Marcadores = 9
  - Señales en mástiles = 10
  - Luces = 11
- Record.user_data: asigna un número a los elementos que pertenecen a una misma vía. Estos elementos son raíles, cables de contacto, catenaria y droppers. Todos los elementos de una misma vía tienen un número comprendido entre una potencia de 10. A los raíles se le asigna una potencia de 10 ($i \cdot 10^x$). Al grupo de cables j de contacto y catenaria con sus droppers pertenecientes a la vía i se les asigna el valor $i \cdot 10^x + j$. El número máximo de cables de contacto-catenaria de cada vía no puede ser superior a 9.
- Record.point_source_id: asigna un número a cada punto en función del grupo al que pertenezcan. Cada grupo tiene un número único.

Si no es capaz de segmentar una nube, guarda una variable con un mensaje de error en la ruta especificada para ello. No se analiza el tipo de error.

Las nubes deben tener georreferencia 3D, intensidad y sellos de tiempo.

La trayectoria debe tener georreferencia 3D y sellos de tiempo sincronizados con los de las nubes.

CÓDIGOS

- Classes
  - Element.m
  - pointCloud_.m
  - Raster2D.m
  - Trajectory.m
  - Voxels.m
- Preprocessing
  - CloudGeneration.m
  - Las2pointCloud_.m
  - PcaFlattering.m
  - SaveParallel.m
  - Sectioning.m
  - TrajectoryGenerate.m
- Railway
  - CablesExtraction.m
  - ClusteringNeighbours.m
  - countElements.m
  - countTime.m
  - DenoisingRails.m
  - DroppersExtraction.m
  - GenerateElements.m
  - Linking.m
  - LinkingBetweenSections.m
  - MastsExtraction.m
  - PcaLocal.m
  - RailsExtraction.m
  - Segmentation.m
  - SignalsExtraction.m
  - TrackSegmentation.m
  - ModifySaveLas.m
  - MerginRails.m
- Models: contiene los archivos .m con las matrices Nx3, que son la nubes de puntos modelo de cada elemento.
- RailwayHeuristicSegmentation.m

1 RAILWAYHEURISTICSEGMENTATION

Entradas

- *pathInTrajectory : char*

  Ruta del archivo de trayectoria.
- *pathIn : char*

  Ruta con las nubes en formato *.las.*
- *pathOut : char*

  Ruta en la que se guardan las salidas.
- *pathErrors : char*

  Rutan en la que se guardan los errores.

Salidas

- *cloud : .las*

  La salida es la nube .las de entrada en la que se modifican los siguientes campos.
  - Record.classification: asigna un número a cada punto en función del tipo de elemento al que pertenezca
    - Rail = 1
    - Catenaria = 2
    - Contacto = 3
    - Dropper = 4
    - Otros cables = 5
    - Mástiles = 6
    - Señales = 7
    - Semáforors = 8
    - Marcadores = 9
    - Señales en mástiles = 10
    - Luces = 11
  - Record.user_data: asigna un número a los elementos que pertenecen a una misma vía. Estos elementos son raíles, cables de contacto, catenaria y droppers. Todos los elementos de una misma vía tienen un número comprendido entre una potencia de 10. A los raíles se le asigna una potencia de 10 ($i \cdot 10^x$). Al grupo de cables j de contacto y catenaria con sus droppers pertenecientes a la vía i se les asigna el valor $i \cdot 10^x + j$. El número máximo de cables de contacto-catenaria de cada vía no puede ser superior a 9.
  - Record.point_source_id: asigna un número a cada punto en función del grupo al que pertenezcan. Cada grupo tiene un número único.

Si no es capaz de segmentar una nube, guarda una variable con un mensaje de error en la ruta especificada para ello. No se analiza el tipo de error.

Las nubes deben tener georreferencia 3D, intensidad y sellos de tiempo.

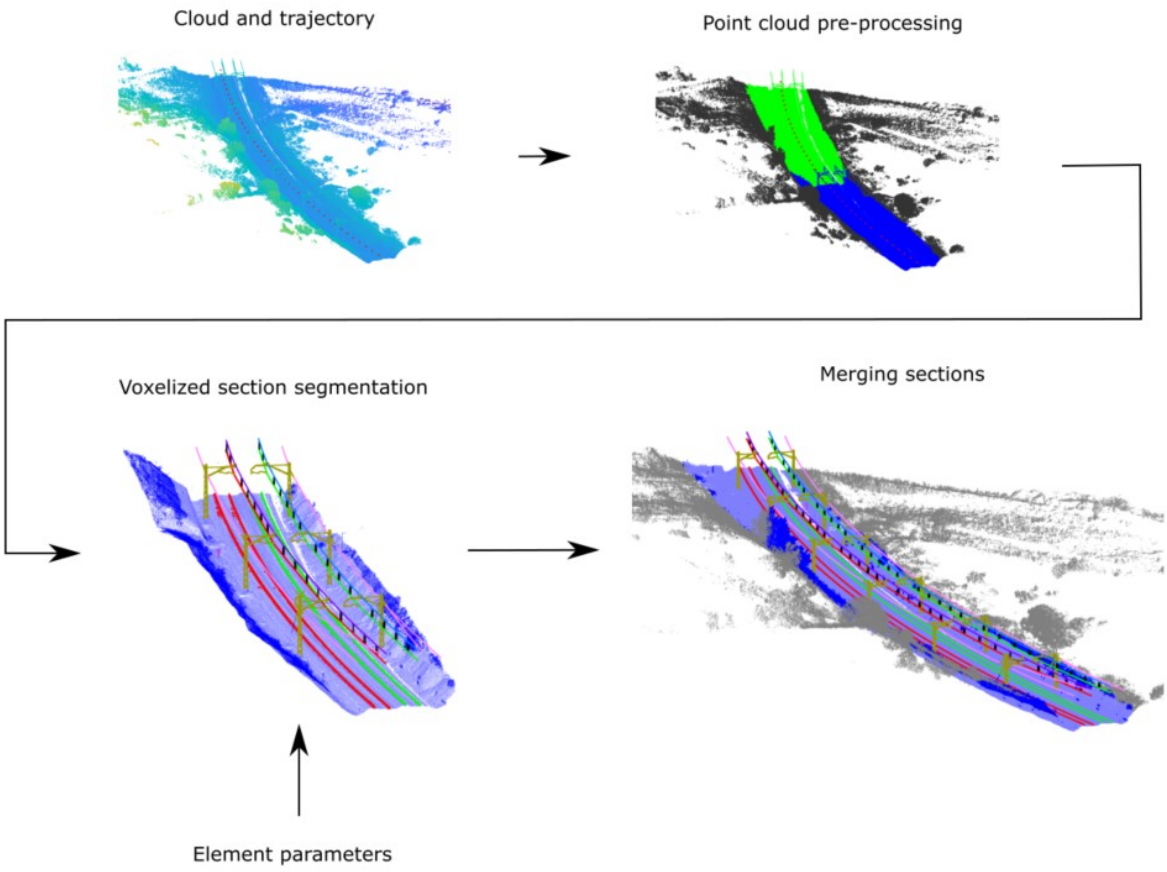
La trayectoria debe tener georreferencia 3D y sellos de tiempo sincronizados con los de las nubes.

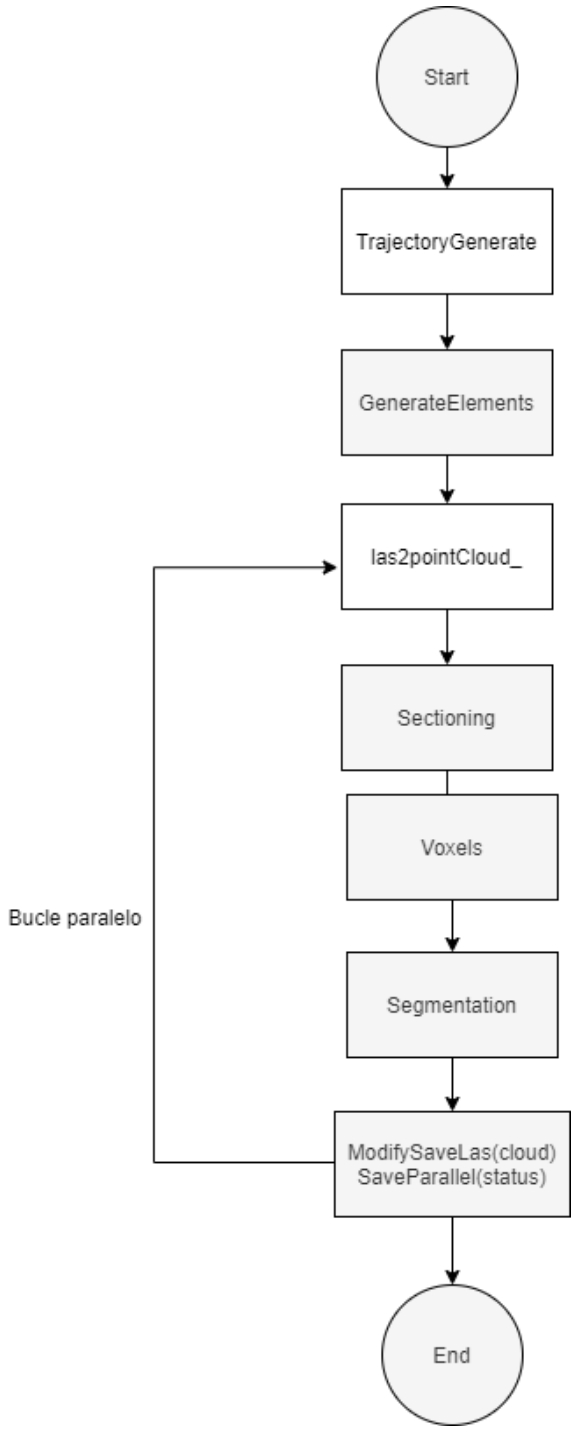
- *status : cell array*

Contiene información sobre el proceso. Las unidades son SI. Tiene la siguiente estructura:

- total. Tiempo de segmentación.
- lasToPointCloud. Tiempo de generación de PointCloud_ a partir de .las
- sectioning. Tiempo de ejecución de *sectioning.*
- voxelize. Tiempo de voxelizado de las secciones.
- segmentation. Tiempo de ejecución de segmentation
- grid. Tamaño de voxelizado de las secciones
- selectSection. Tiempo de selección de las secciones voxelizadas de la nube voxelizada.
- Track. Tiempo de extracción del suelo.
- Rails. Tiempo de extracción de los raíles.
- Directions. Tiempo de ejecución de *PcaLocal*.
- Mast. Tiempo de extracción de los mástiles
- Droppers. Tiempo de extracción de los droppers.
- Signs. Tiempo de extracción de las señales.
- indexVx. Tiempo de conversión de índices de secciones a índices en la nube voxelizada.
- CloudIds. Tiempo de conversión de los índices de la nube voxelizada a la nube original.
- Length. Longitud de la nube.
- Section. Celdas con el tiempo empleado en segmentar cada sección de la nube.
- mergeSections. Tiempo empleado en unir las secciones.

Descripción





Es el código principal. Las entradas se especifican en las primeras líneas del código.

Se genera el objeto *Trajectory* a partir del archivo de la trayectoria. Después se genera la variable model que contiene objetos *Element* con las especificaciones de los elementos que se segmentan en este código. Algunos de ellos tienen una nube de puntos cargada en disco. Esta nube debe estar en formato .m y ser una matriz nx3.

En el bucle paralelo se genera el objeto *PointCloud_* a partir del .las de cada nube, se secciona, voxeliza, se segmenta y se guarda.

La función *SaveParallel* se usa porque no se puede usar directamente la función de Matlab save() dentro de un bucle paralelo.

2 TRAJECTORYGENERATE

Entradas

- *pathIn : char*

  Ruta del archivo en ASCII de trayectoria
- *wantSave : logical*

Si se quiere o no guardar en disco el objeto
- *pathOut : char*

Ruta de guardado del objeto Trajectory. Si no se le indica ninguna se usa la ruta del archivo de entrada.
- *orthometricToElipsoidal : logical*

Si se quiere convertir a coordenadas elipsoidales en caso de que las coordenadas del archivo estén en ortométricas.

Salidas

- *traj : Trajectory*

  Objeto de la clase *Trajectory*

Descripción

Lee el archivo .txt en formato ASCII usando la función de Matlab textscan() indicándole las especificaciones de un archivo ASCII.

Si así se indica, convierte las coordenadas ortométricas a elipsoidales, calculando la diferencia entre ambas con la función de Matlab geoidheight().

Genera el objeto Trayectoria.

Si así se indica, se guarda el objeto.

3 GENERATEELEMENT

Entradas

- *pathModels : char*

Ruta a las nubes de puntos de los modelos. No todos los modelos tienen una nube de puntos. Estas nubes son únicamente la parte de arriba de las señales para que no influya el efecto del mástil.

Salidas

- *Model : cell array de Element*

Estructura que contiene los modeles de los elementos a extraer como objetos *Element*.

Descripción

Este código genera los modelos como objetos *Element* de los elementos que se van a extraer. Algunos de estos modelos se usan para comparar sus atributos con las características de puntos agrupados en el proceso de segmentación para clasificarlos. Otros contienen atributos que se usan como parámetro de algunos procesos de segmentación.

Algunos de estos modelos tienen como atributo una nube de puntos modelo. Estas nubes se cargan a partir de la ruta de entrada.

En este caso, las nubes que se cagan son 3 tipos de semáforos, 2 tipos de marcadores (stoneSignal y 3) y un tipo de señal que parece una luz (stoneSignal2).

Los modelos generados son:

- bigSignalModel. Señales reflectantes.
- trafficLightModel. Semáforo
- trafficLightModel2. Semáforo
- trafficLightModel3. Semáforo
- stoneSignalModel. Marcador
- stoneSignalModel3. Marcador
- light. Luz
- mastNoBracketModel. Mástil sin ménsula
- cableModel. Cable de catenaria, contacto u otro.
- dropperModel. Droppers.
- railModel. Raíl.
- railPairModel. Pareja de raíles.

4 LAS2POINTCLOUD_

Entradas

- *lasFile : char*
  Ruta de la nube .las

Salidas

- *cloud : PointCloud_*
  Nube de puntos.

Descripción

Lee el archivo usando la función de Lasread() de la librería LASio y genera el objeto PointCloud_.

5 SECTIONING

Entradas

- *Cloud : pointCloud_*

Nube que va a ser seccionada

- *Traj : trajectory*

Trayectoria que incluye la parte relativa a *cloud*. Puede ser únicamente la parte de *cloud* o la trayectoria total.

Salidas

- *Cloud : pointCloud_*

  Nube formada por todas las secciones de nube unidas.
- *trajCloud : Trajectory*

  Trayectoria formada por todas las secciones de trayectoria unidas.
- *Sections : cell array*
  - *Sections.traj : cell array*

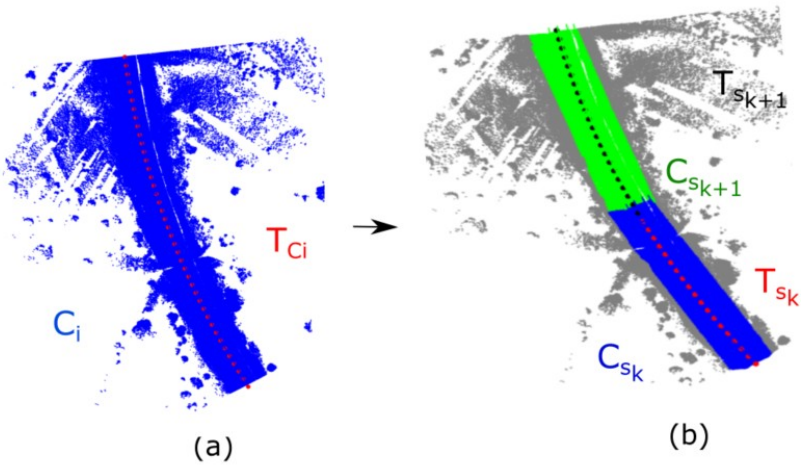
  Cada celda contiene los índices de *trajCloud* de cada sección.
  - *Sections.cloud : cell array*

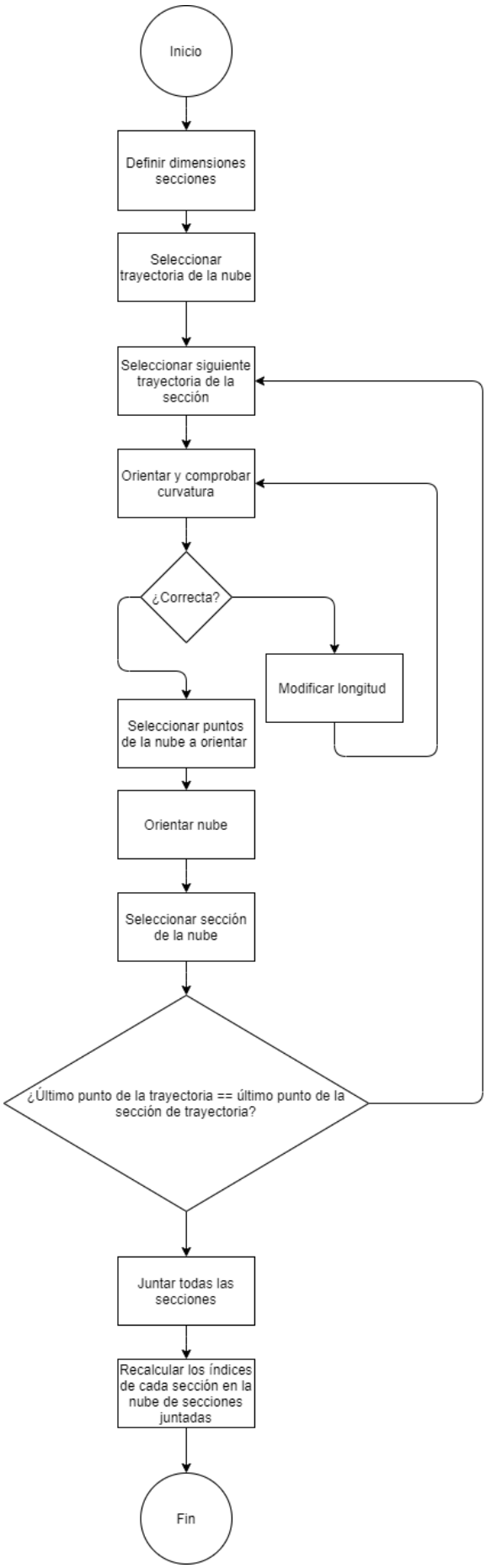
  Cada celda contiene los índices de *cloud* (salida) de cada sección.
- *idxCloud*

  Índices de los puntos de *cloud* (entrada) que están en *cloud* (salida).
- *idxTraj*

  Índices de los puntos de *traj* que están en *trajCloud*.

Descripción




Inicio
Definir dimensiones secciones
Seleccionar trayectoria de la nube
Seleccionar siguiente trayectoria de la sección
Orientar y comprobar curvatura
¿Correcta?
Modificar longitud
Seleccionar puntos de la nube a orientar
Orientar nube
Seleccionar sección de la nube
¿Último punto de la trayectoria == último punto de la sección de trayectoria?
Juntar todas las secciones
Recalcular los índices de cada sección en la nube de secciones juntadas
Fin

La nube, usando la trayectoria, se secciona. El objetivo es preparar la nube para el proceso de segmentación, limitando la longitud y la curvatura máxima, y retirando los puntos lejanos a la trayectoria.

Las secciones están ordenadas usando el sello de tiempo. Primero, se determina una sección de trayectoria empezando por el último punto incluido en la sección anterior. Esta sección de trayectoria tiene una curvatura menor que la *maxCurvature* y una longitud menor que *step*. Usando esta trayectoria, se seleccionan los puntos de la nube relativas a esta, no incluyendo los puntos a una distancia de la trayectoria mayor que *width*. A mayores, se incluyen puntos que pertenecerán al comienzo de la sección siguiente hasta una distancia igual a *overlap*. Esto se hace para mejorar el proceso de segmentación ya que, si algún elemento cuadra en medio de 2 secciones, al menos en 1 de ellas estará completa.

Todas las secciones se guardan en una única nube para no usar la original, ya que tiene puntos que no se van a analizar y así se libera espacio. No obstante, para poder guardar los resultados referidos a la nube original, se guardan los índices de la nueva nube con respecto a la original.

5.1 Parameters for sections

Se definen los parámetros de las secciones.

- *Step* : largo máximo de la sección. Será menor si la curva en ella es demasiada
- *maxCurvature* : limita la curvatura. Es el rango máximo en la dirección perpendicular a la trayectoria
- *width* : ancho de la sección.
- *Overlap* : solapamiento con la sección siguiente.

5.2 Trajectory

Se selecciona la parte de *traj* que pertenece a *cloud*. Para ello primero se cogen los puntos de *traj* dentro del rango de sellos de tiempo de *cloud*. Una vez seleccionados estos, se cogen los que se encuentren dentro de la caja limitadora de *cloud*. Por último, se seleccionan los puntos de *cloud* más cercanos al primero y al último de los puntos de *traj* seleccionados. Después se buscan los puntos de *traj* más cercanos a los seleccionados en *cloud*. Estos puntos son el primero y el último de la trayectoria correspondiente a la nube, llamada *trajCloud*.

Quizás con el último paso sin aplicar los anteriores sería suficiente, pero este último paso fue algo que se añadió después y no se borraron los métodos anteriores.

5.3 Sections

En un bucle while se secciona la trayectoria y en base a esta se secciona la nube hasta que toda la trayectoria es seccionada.

Primero se selecciona la trayectoria. El primer punto de la trayectoria es el siguiente al último punto de la sección anterior. Por distancia, se calcula el último punto de la trayectoria de esta sección. Los puntos entre el primero y el último se orientan usando *PcaFlatterring*. Se comprueba su curvatura. Si es demasiada, se hace más corta la trayectoria.

En el caso de que no se encuentre ningún punto como último de la trayectoria (porque se acaba la trayectoria), o este es el último de la trayectoria, esta sección será la última. En este caso se sigue el mismo proceso, pero a la inversa. Es decir, se coge como primer punto de la trayectoria de la sección el último de la nube, y como último una anterior, haciendo los mismos cálculos de distancia y curvatura.

A partir de la sección de trayectoria se secciona la nube. Para ello la nube se orienta usando la misma orientación que para la sección de trayectoria. Para no orientar todos los puntos de la nube, se cogen únicamente puntos que tengan un sello de tiempo menor que el sello de tiempo

máximo de la trayectoria más el rango, teniendo así cierto margen. Además, se cogen únicamente puntos que se encuentren en la variable *restPointsCloud*, para descartar puntos de secciones anteriores considerando el solapamiento.

Los puntos seleccionados se orientan. Los puntos de esta sección son aquellos dentro del rango en Y definido por step, y dentro del rango en X de la trayectoria más el solapamiento.

Por último, se actualiza la variable *restPointsCloud*, eliminando todos los puntos que con X menor que el máximo X de la sección de trayectoria.

Si es la última sección, no se tiene en cuenta la variable *restPointsCloud*, seleccionando los puntos con un sello de tiempo mayor que el mínimo de la sección de la trayectoria menos el rango de esta.

5.4 Selecting only the idx of the cloud that are in any section

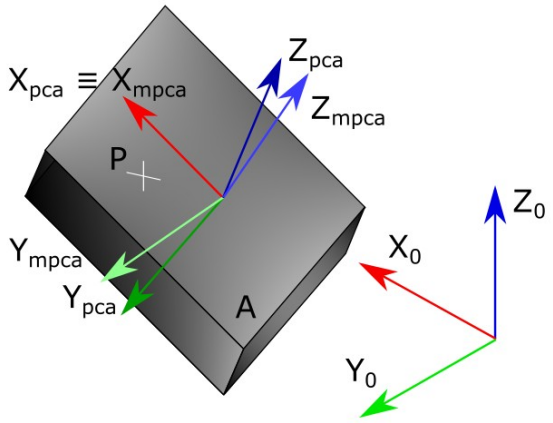
Para no trabajar con toda la nube, se genera una nueva nube únicamente con los puntos que se encuentren en alguna sección. Esto se hace uniendo los índices de puntos de todas las secciones y seleccionando estos puntos en la nube original. Se hace lo mismo con la trayectoria.

Como estos índices estaban referidos a la nube original, hay que recalcular sus índices en la nueva nube. Lo mismo con la trayectoria.

6 PCAFLATTERING

Esta función sirve para modificar el resultado de los autovectores de PCA. Esto se hace para conseguir que el eje Ypca esté en el plano XoYo. Si el elemento al que se aplica PCA es muy lineal, o tiene irregularidades que no se quiere que afecten, o simplemente se quiere mantener su orientación vertical, se usa esta función. De esta forma, el objeto se orienta según su dirección principal, sin escorarse. En el caso de un fragmento de raíl, que es más alto que ancho, si se orienta con PCA este se tumba o queda rotado. Con esta función, se orienta con respecto a su trayectoria, pero se mantiene vertical.

Esto se consigue aplicando PCA. El primer autovector es el calculado por PCA. La dirección del segundo es la proyección del segundo autovector calculado por PCA en el plano XoYo. El tercero es ortonormal a los 2 y con su componente Z positiva, para que apunte hacia arriba.



7 SEGMENTATION

Entradas

- *vx : Voxels*
  Nube de todas las secciones unidas voxelizada
- *Traj : trajetory*
  Trayectoria de todas las secciones unidas
- *Sections : cell array*
  Contiene los índices de los puntos de cada sección en la nube de secciones unidas (la nube que está voxelizada en vx) y en traj
- *idxOriginal: Nx1 numeric*
  Índices de los puntos de la nube original en la nube de secciones unidas.
- *Model : cell array of elements*
  Variable con los modelos de los elements a extraer
- *Status: cell array*
  Variables de tiempo de ejecución

Salidas

- *Components*: cell

Contiene un conjunto de celdas. Cada celda pertenece a un tipo de elemento, y dentro de cada celda hay tantas celdas como elementos de ese tipo hay presentes en la nube. Dentro de ellas están los índices de los puntos que los forman. Su estructura es la siguiente:

- Signs
  - Light (no desarrollado)
  - Big
  - trafficLight
  - stone
  - inMast
- bracketTunnel (no desarrollado)
- track
- notTrack
- wall
- roof
- rails
  - pareja 1
    - izquiedo
    - derecho
  - pareja 2
- masts
- cables
  - pairs
    - de rails pareja 1
      - cables 1
        - 1 (pareja)

        - 2 (catenary)
            - cables 2
        - de rails pareja 2
    - others
        - others 1
        - others 2
- droppers
- evaluated

Hasta la función *mergingRails*, los raíles, cables de contacto y catenaria y droppers siguen el mismo orden. Es decir, la primera celda de railes 1 (izquierda) y 2 (derecha) son los raíles que tienen encima la primera pareja de cables 1 (contacto) y 2 (catenaria). Además, la primera celda de droppers tiene las celdas de los droppers de esos cables. La segunda pareja de raíles corresponde a la segunda pareja contacto-catenaria, y sus respectivos droppers del segundo grupo de droppers.

A partir de la función merginRails, la estructura varía. Hasta este paso, las dependencias se hacían a partir de los cables de contacto y catenaria. A partir de esa función, dependen de los raíles. Esto es necesario para evitar que un mismo raíl se clasifique como varios si se produce un cambio de cables contacto-catenaria. En la nueva estructura, la primera celda de railes 1 (izquierda) y 2 (derecha) son los raíles que tienen encima las parejas en la primera celda de cables de contacto y catenaria. Dentro de esa primera celda, hay celdas con 2 grupos cada una, uno perteneciente a los cables de contacto y la otra a los de catenaria. Los droppers están organizados de la misma forma que los cables de contacto-catenaria.

En evaluated están los índices que pertenecen a todas las secciones de la nube. Los puntos que no están en esa variable no han sido analizados, ya que fueron descartados en el seccionamiento.

- *Status*: explicado en RailwayHeuristicSegmentation

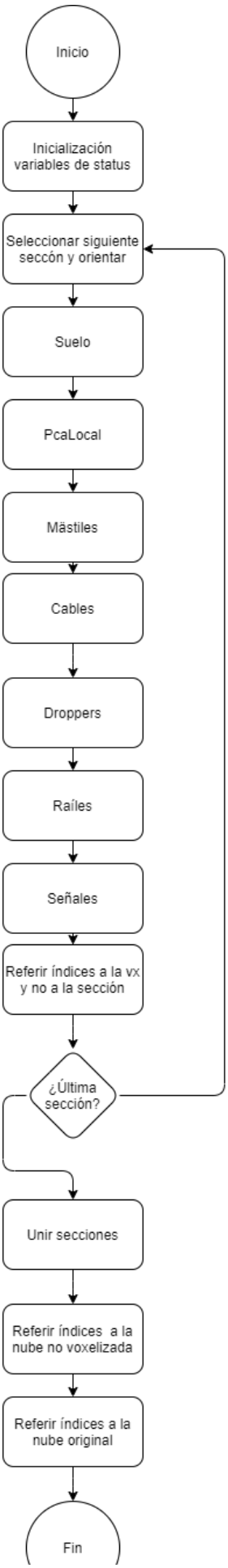
Descripción

A partir de la nube voxelizada, sección a sección, se segmentan sus componentes de forma secuencial.

Una vez segementadas todas las secciones, se unen, comprobando si hay elementos repetidos y uniendo elementos que son el mismo.

Aunque el proceso se realice sobre la nube de puntos voxelizada, los índices finales están referidos a la nube de puntos original.

Se recuerda que la nube que es voxelizada no es la nube original. Es la nube de todas las secciones unidas.


Inicio
Inicialización variables de status
Seleccionar siguiente sección y orientar
Suelo
PcaLocal
Mästiles
Cables
Droppers
Raíles
Señales
Referir índices a la vx y no a la sección
¿Última sección?
Unir secciones
Referir índices a la nube no voxelizada
Referir índices a la nube original
Fin

7.1 Selecting the section and orienting

A partir de los índices en *sections*, se selecciona *trajSec* y *vxSec*, que son la trayectoria y la nube voxelizada de la sección a segmentar. En el caso de la trayectoria es inmediato. En el caso de la nube voxelizada, como los índices están referidos a la nube sin voxelizar, se seleccionan los vóxeles que tengan al menos un punto de esta sección. En *vx.parent_idx* se encuentran los índices de la nube sin voxelizar en cada voxel.

Tanto la *trajSec* como la *vxSec* se orientan usando *PcaFlattering* aplicado a *trajSec*.

7.2 Segmentation

Son los procesos que van desde la segmentación del suelo hasta la segmentación de señales. Cada proceso utiliza una función. En la descripción de cada una de ellas se explica su funcionamiento. La secuencia de ejecución es importante ya que es en cadena.

7.3 Index in vx, not in vxSec

Los índices extraídos en el proceso de segmentación están calculados en *vxSec*. Usando *idxSection*, que contiene los índices de *vxSec* en *vx*, se obtienen los índices en *vx*.

7.4 Merging components of different sections

Cada sección se segmentó de forma independiente, guardando el resultado en *componentsVxSec{}*, que contiene tantas celdas como secciones. En ella están los índices de cada elemento extraído ya referidos a vx.

El resultado de unir las secciones se guarda en *componentsVx*, que sigue la misma estructura que componetnsVxSec.

Todos los índices de *componentsVxSec{1}*, se copian en *componentsVx*.

Por orden, se comparan los elementos del mismo tipo en una sección con la anterior.

Los elementos puntuales y no dependentes, es decir, todos menos los cables, raíles y droppers.

Se va recorriendo cada elemento en la sección estudiada, mirando que ninguno de sus índices esté en algún elemento del mismo tipo en la sección anterior. Si no lo está se añade el elemento en *componentsVx*.

Los elementos continuos a unir son los cables, y para ello se diferencia entre catenaria-contacto, y otros cables. Los raíles y droppers se tratan como elementos continuos del par catenaria-contacto.

La variable *cablesPairsCluster* es un vector que contiene tanto elementos como parejas catenaria-contacto hay en la sección anterior. Este vector asigna a cada pareja un índice el cual indica a que pareja pertenece cada uno en *componentsVx*.

Si en la sección anterior no hay contacto-catenaria, las parejas en esta sección se añaden directamente como parejas nuevas en *componentsVx*, y se le asignan sus índices correspondientes en la variable *cablesPairsCluster*.

Si en la sección anterior hay contacto-catenaria y en esta también, se usa la función *LinkingBetweenSections* para calcular la variable *newCluster*. Esta variable es un vector de tantos elementos como contacto-catenaria hay en esta sección. Este vector indica con qué pareja en *componentsVx* se conectan las parejas de esta sección. Si *newCluster(1)* = 5, indica el la pareja 1 en esta sección es la continuación de la pareja 5 en *componentsVx*, y por consiguiente la continuación de la pareja en la sección anterior donde *cablesPairCluster(x)* = 5. Si *newCluster(y)* = 0, esta pareja de cables no es la continuación de ninguna anterior.

Una vez calculado a qué pareja pertenece cada una, se añaden sus índices en donde corresponde en *componentsVx*. Si es una pareja nueva, se añaden todos como una pareja nueva,

y se modifica el valor de *newCluster* de esa pareja, asignándole el que corresponda. Si no es nueva, se añaden todos menos los que están en la sección anterior para no repetir índices.

Los raíles y los droppers se añaden al mismo grupo al que pertenecen los raíles y droppers dependientes de su pareja de cables. En el caso de los raíles se usa el mismo procedimiento que en los cables para no repetir índices. En el caso de los droppers, se usa el mismo procedimiento que en los elementos puntuales, comparándolos solamente con los droppers dependientes de sus mismos cables.

La variable *cablesPairsCluster* se actualiza, igualándola a *newCluster*.

Los otros cables siguen el mismo procedimiento, usando *cablesOtherCluster en vez de cablesPairsCluster*.

7.5 Index in cloud, not in vx

Se calcula la variable *components* a partir de *componentsVx*. *Components* contiene los mismos elementos que *componentsVx*, pero con los índices referidos a la nube que fue voxelizada en *vx*. De esta forma, en cada elemento, cada índice de cada voxel es sustituido por todos los índices en la nube padre contenidos en ese voxel. En este proceso se aplica la función *denoisingRails*. En la nube voxelizada, debido al tamaño del voxel, no es posible diferenciar con suficiente exactitud qué voxeles pertenecen al balasto y cuales a los raíles. Por eso, en este paso a índices en la nube no voxelizada, se aplica esta función para eliminar los puntos que no pertenecen a los raíles.

7.6 Index in the original cloud

Se modifica la variable *components* usando *idxOriginal*. De esta forma los índices, en vez de estar referidos a la nube padre de *vx*, están referidos a la nube original.

8 TRACKSEGMENTATION

Entradas

- *Vx : Voxels*
  Nube voxelizada y orientada. En el código se le pasa *vxSec*
- *Grid : numeric*
  Tamaño del voxel para la extracción de suelo y de las paredes
- *subSectionsWidth : numeric*
  Ancho de las secciones en la extracción de paredes
- *distWall : numeric*
  Marge en la extracción de las paredes.
- *Traj : trajectory*
  Trayectoria correspondiente a la nube

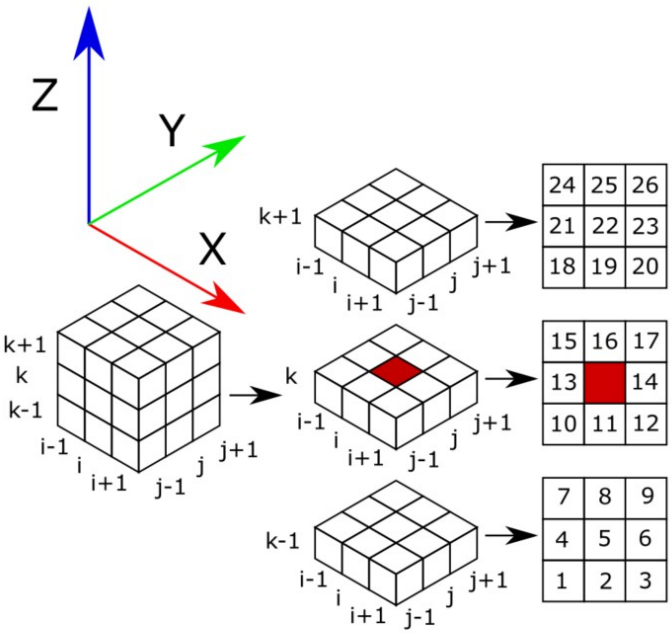
Salidas

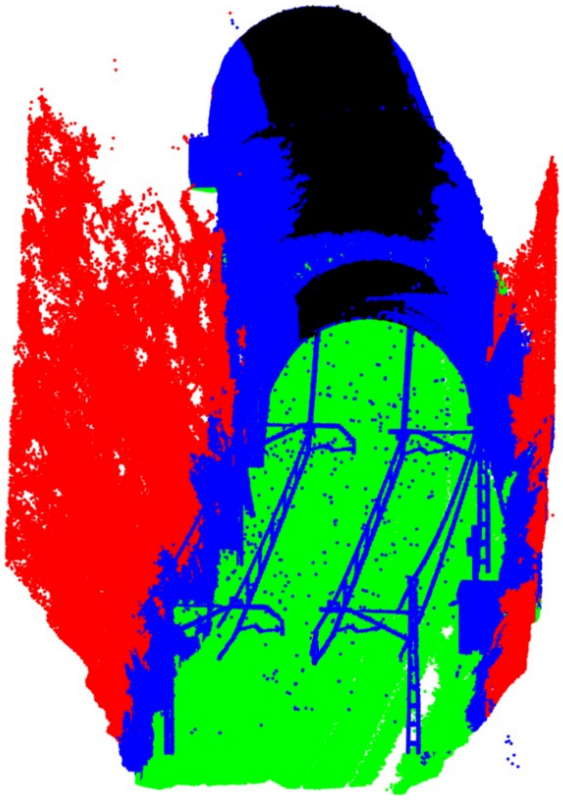
- *Components.track: numeric*
  Indices en *vx* del suelo
- *Components.notTrack : numeric*
  Indices en *vx* que no están ni en *components.track* ni en *compoents.roof*
- *Components.wall : numeric*
  Índices en *vx* de muros o puntos lejanos, son los que están más allá del suelo
- *Components.wall2 : numeric*
  Índices en *vx* igual que *components.wall* pero usando *distWall*
- *Components.roof : numeric*
  Índices en *vx* del techo

Descripción

Este método extrae de una nube voxelizada orientada, primero los índices del suelo (verde), después los muros o voxeles remotos (rojo), y por último vóxeles pertenecientes a algún techo (blanco).

La extracción del suelo sirve para extraer los raíles de él y para aislar los elementos que no están en él, como las señales. La extracción de los muros sirve para no buscar en ellos ciertos elementos. La del techo para hacer una extracción buena de los cables en los túneles, eliminando el efecto de este.





El proceso de extracción del suelo está basado en la vecindad de sus vóxeles. Para que esto funcione, la nube debe de estar voxelizada a un tamaño de voxel mayor que los elementos que queremos que pertenezca al suelo, en este caso, mayor que el alto de los raíles.

Primero, se generar *vxBigVoxels*, voxelizando *vx* con el tamaño de voxel *grid*. *vxBigVoxels* se usa para la extracción del suelo y de los muros.

En *vxBigVoxels*, se seleccionan los vóxeles que no tienen ni vecino 5 ni 22. Vóxeles que no tengan vecino ni justo encima ni justo debajo. Estos voxeles se agrupan usando *ClusteringNeighbors*, y se eliminan los grupos con pocos vóxeles. Seguidamente, todos los vóxeles en grupos cuya Z media sea inferior a la Z media de la trayectoria son segmentados

como suelo. A mayores, los voxeles que tengan algún vecino entre el 10 y el 17 segmentando como suelo también se segmentan como suelo.

Para segmentar los muros, la nube se secciona por planos verticales perpendiculares a X de ancho *subSectionsWidth*. En cada sección se calcula la Y mínima y la Y máxima del suelo en ella. Los puntos más allá de estos límites se segmentan como *wall*. Esos mismos límites, añadiéndole *distWall*, se usan como límites a partir de los cuales los vóxeles más allá de ellos se segmentan como *wall2*.

Como estos índices son calculados en *vxBigVoxels*, se recalculan los índices en *vx*.

Para la extracción de techo, se aplica la función PcaLocal a los vóxeles de *vx* que no estén ni en el suelo ni en los muros. Los vóxeles con una normal vertical y un con un primer autovalor pequeño, lo que indica que pertenece a un elemento plano, se agrupan usando *ClusteringNeighbors*. Los vóxeles en grupos suficientemente grandes se segmentan como techo.

9 PCALOCAL

Entradas

- *Vx : Voxels*

  Nube de puntos voxelizada. En este caso es *vxUp*, que son los vóxeles pertenecientes a notTrack en vxSec.

- *maxDistance : numeric.*

  Distancia que define la vecindad de un voxel.

- *Mode : numeric*

  Define el modo de funcionamiento. 1 usando únicamente maxDistance. 2 usando maxDistance y la vecindad de Voxels.

Salidas

- *Directions.eigenvectors : Nx9 numeric.*

  Contiene los autovectores de aplicar PCA a los N Vóxeles con su vecindad.

- *Directions.eigenvalues : Nx3 numeric.*

  Contiene los autovalores de aplicar PCA a los N Vóxeles con su vecindad.

Descripción

Esta función se usa para estudiar el entorno de cada voxel y así poder determinar su forman parte de un elemento vertical, plano, lineal, longitudinal, etc. En *trackSegmentation* se usa para extraer el techo. En *segmentation* se aplica a los puntos en *notTrack*. Esto se hace porque en el suelo solamente se van a extraer raíles y no se usa esta función. El motivo de utilizar todos los puntos en *notTrack*, incluyendo los pertenecientes a *componentsSecVx.wall* es porque si se elimina, se pueden obtener resultados extraños en vóxeles que pertenecen a elementos que tienen una parte en *componentsSecVx.wall*.

Se aplica con 2 valores de *maxDistance*. El mayor de ellos se usa para la extracción de mástiles. El otro para el resto de las extracciones. Además, solamente se usa el modo 1.

Primero, se inicializan las variables de salida.

Después, se aplica la función de Matlab rangesearh(). Como resultado, para cada voxel, se calculan los índices de los vóxeles que se encuentran a una distancia menor que *maxDistance*.

Se recorre en un bucle for todos los puntos de la nube, seleccionando cada punto y todos los puntos a una distancia menor que *maxDistance*. A estos puntos se le aplica PCA y se guarda su resultado. Se usa pcacov(cov(nube)) en vez de pca(nube) porque es muchísimo más rápido.

En el código, *components.directionsAll* es el resultado usando la mayor distancia y *components.directionsUp* la menor.

10 MASTSEXTRACTION

Entradas

- *Vx : Voxels*
  Nube voxelizada *vxSec*
- *Components : cell*
  Estructura *compoentsVxSec*
- *mastNoBracketModel : Element*
  Modelo de los mástiles

Salidas

- *Components.roughMasts : cell 1 x numMasts*
  Celdas con los índices de cada mástil sin su ménsula
- *Components.masts : ceel 1 xnumMasts*
  Celdas con los índices de cada mástil

Descripción

El objetivo de esta función es segmentar los mástiles. Primero, se buscan los mástiles sin su ménsula y después se le añade. Para ello, se utiliza el resultado de *PcaLocal* con la mayor vecindad (*compoents.directionsAll*), que fue aplicado en los puntos en *components.notTrack*. Se usa la vecindad grande ya que la pequeña está pensada para cables y droppers, que son elementos más finos. Se seleccionan puntos con una componente Z elevada y una componente X e Y pequeña en el primer autovector. A estos puntos se les añaden sus vecinos. Esto es necesario porque no se cogen todos los vóxeles del mástil y al agruparlos podrían romperse en 2 grupos un mismo mástil. Aún así, siguen sin quedar completos.

Los vóxeles se agrupan usando la función de Matlab DBSCAN en vez de *ClusteringNeighbors* precisamente porque faltan vóxeles en el mástil. Se aplica el método *CheckElement* comparando cada grupo con el modelo del mástil sin ménsula *mastNoBracketModel*, descartando los grupos que no superan la comparación.

Como último paso de la extracción de mástiles sin ménsula, se añaden los vóxeles que le faltan. Para ello, se calcula el centro de cada grupo. Medido sobre su centro, se seleccionan todos los puntos dentro de los límites en X y en Y de las dimensiones de un mástil definidos por *mastNoBracketModel*. La restricción en Z es los puntos tengan una coordenada Z mayor a 2 m del mínimo del grupo, evitando así coger puntos del suelo o de pared muy cercana al suelo. Después, estos puntos se agrupan usando DBSCAN y se busca el grupo con mayor número de vóxeles en el grupo original. Los vóxeles en este grupo se añaden a los de la función original, evitando repeticiones, completando así el proceso de extracción de *mastNoBrackets*.

Para añadir las ménsulas, se seleccionan todos los vóxeles en *compoents.notTrack* con una baja componente X en su primer autovector. De esta forma se cogen vóxeles de las ménsulas, de otros elementos y también de los propios mástiles. A mayores, se descartan todos aquellos que en *components.wall2* y con una Z menor que la de la trayectoria (menor que 0), ya que se están

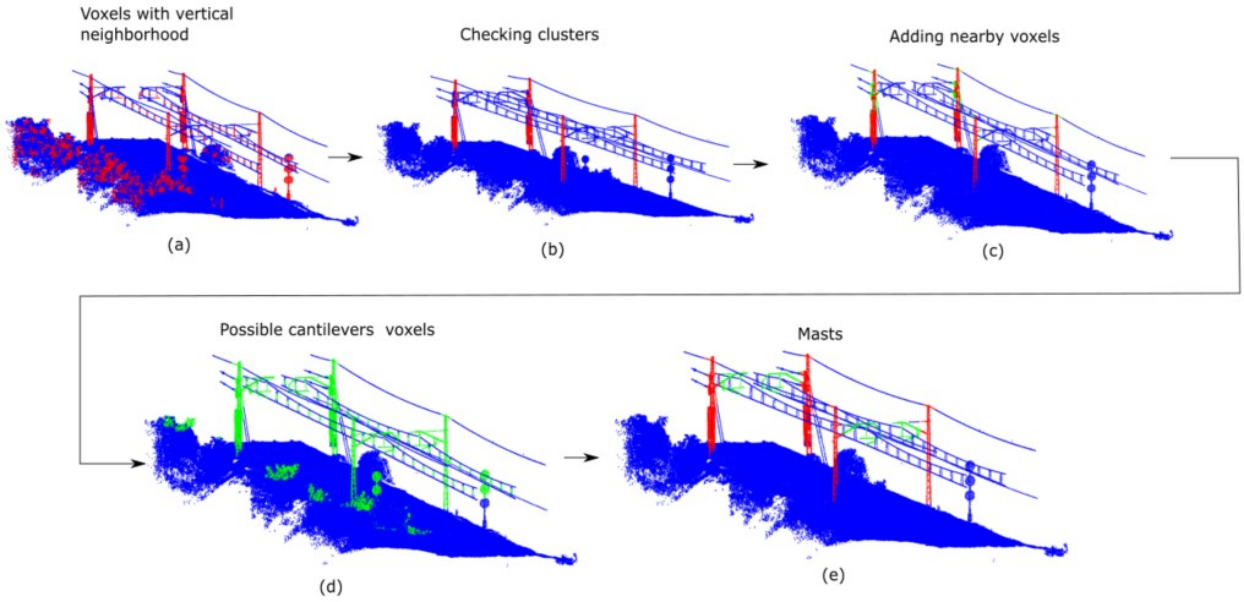
buscando ménsulas. A estos puntos se le añaden sus vecinos, borrando de nuevo los vóxeles en *compoents.wall2*.

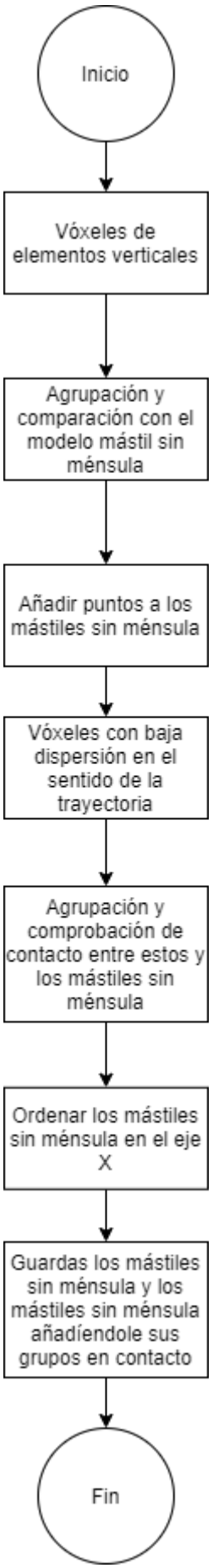
Los vóxeles seleccionados se agrupan y se comprueba con qué *mastNoBrackets* comparten vóxeles.

Para favorecer procesos de extracción de otros elementos, los *mastNoBrackets* se ordenan en X.

Por último, se guardan *mastNoBrackts* en *components.roughMasts*, y en *components.masts* se guardan *mastNoBrackets* con sus grupos en contacto, que son sus ménsulas.

En *RailsExtraction* se comprueba si los mástiles están en contacto con algún cable. De no estarlo no se segmenta como mástil.




Inicio
Vóxeles de elementos verticales
Agrupación y comparación con el modelo mástil sin ménsula
Añadir puntos a los mástiles sin ménsula
Vóxeles con baja dispersión en el sentido de la trayectoria
Agrupación y comprobación de contacto entre estos y los mástiles sin ménsula
Ordenar los mástiles sin ménsula en el eje X
Guardas los mástiles sin ménsula y los mástiles sin ménsula añadíendole sus grupos en contacto
Fin

11 CABLESEXTRACTION

Entradas

- *Vx : Voxels*
  Nube de puntos voxelizada *vxSec*
- *Components : cell*
  Estructura *componentsVxSec*
- *CableModel : Element*
  Modelo de los cables
- *Step : numeric*
  Ancho de las secciones usadas en la función Linking para unir los vóxeles de los cables
- *keeppingSeeds : numeric*
  Parámetro en la función Linking para unir los vóxeles de los cables
- *Axis : vector*
  Dimensiones tenidas en cuenta en *Linking*
- *minLength : numeric*
  mínima longitud de un cable
- *marginSearch : numeric*
  Margen de búsqueda bajo y sobre un cable en el proceso de clasificación.

Salidas

- *Components.cables.others : cell 1x num cables other*
  Celdas con los índices de los cables clasificados como otros
- *Components.cables.pairs : cell 1 n num cables pairs*
  Celdas con 2 celdas cada una, la primera con los índices de los vóxeles de una catenaria y la segunda con las de su contacto.

Descripción

El objetivo de esta función es la de segmentar y clasificar los cables.

Primero, se seleccionan vóxeles con una dispersión en el sentido de la trayectoria. Para ello, se utiliza el resultado de *PcaLocal* con la menor vecindad (*compoents.directionsUp*), que fue aplicado en los puntos en *components.notTrack*. Se seleccionan los puntos que una componente X mayor que 0.5 en su primer autovector, eliminando los mástiles, y con una coordenada Z mayor que la de la trayectoria.

Estos vóxeles se agrupan usando la función *Linking*.

En cada grupo se analiza la longitud, la linealidad (media del primer autovalor de los vóxeles del grupo), y densidad (vóxeles/longitud del cable). Los cables que no cumplan con los parámetros mínimos son descartados.

Clasificación. Los cables se clasifican en contacto, catenario u otros. Para ser clasificado como contacto o catenaria debe tener otro cable encima o debajo, sino se clasifica como otros. El proceso consiste en rasterizar los grupos de los cables desde una vista de pájaro, calculando qué pixeles pertenecen a cada cable. Después, los píxeles de cada cable son comparados con el resto de cables buscando píxeles compartidos. El cable que se está comparando con los demás se hace más ancho, añadiéndole en cada fila de la imagen un pixel a la derecha y otro a la izquierda. Se le añade uno porque el tamaño del pixel es *marginSearch**2/3. El cable que más % de pixeles comparta con y si este % es superior al 50% es asignado como su pareja. Analizando la Z media de los vóxeles de cada cable únicamente presentes en los píxeles compartidos, se determina si el cable estudiado es contacto o catenaria en la variable *classification*(i,1) y en *classification*(i,2) se guarda el índice del grupo de su pareja. El % se mide sobre el cable que menos píxeles tiene.

Los cables a los que se le asignó la misma pareja se unen como un único cable.

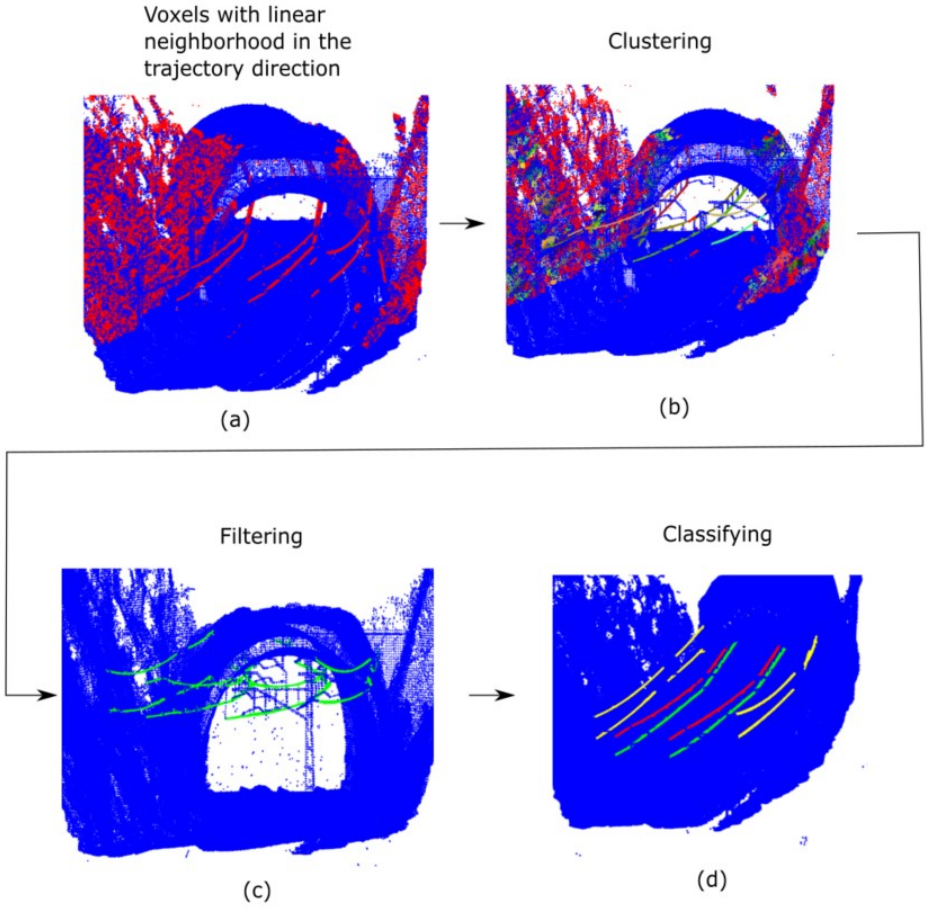
Los cables clasificados como otros, que no tienen pareja, deben estar en contacto con algún mástil. De no ser así se descartan. Este contacto se analiza por distancia.

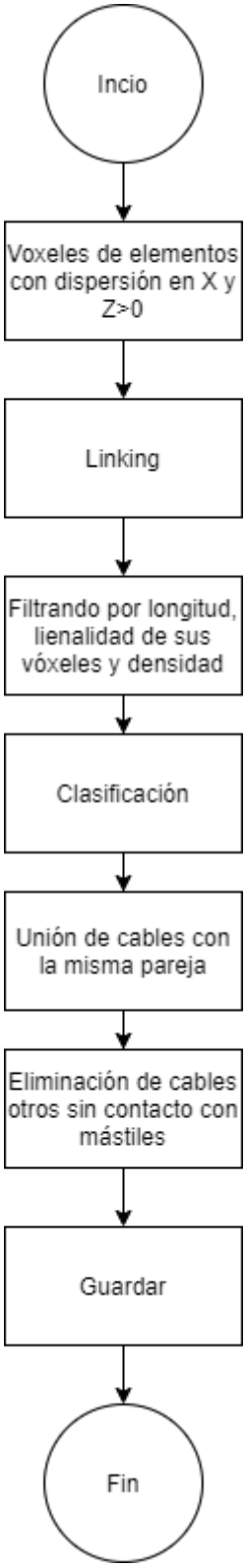
Por último, se guardan los cables clasificados como otros en *components.cables.others*, y los clasificados como contacto y catenaria en *components.cables.pairs*. En cada pareja, el elemento {1} es el contacto y el {2} es la catenaria.

En la función *droppersExtraction*, si una pareja de cables no tiene droppers entre ellos, se reclasifican como otros cables.

En la función *railsExtraction*, si una pareja de cables no tiene raíles asociados, se reclasifican como otros cables.

En la función *railsExtraction*, como se han reclasificado cables como otros se repite la comprobación de contacto entre otros cables y algún mástil, eliminando los cables que no estén en contacto con ninguno.




Incio
Voxeles de elementos con dispersión en X y Z>0
Linking
Filtrando por longitud, lienalidad de sus vóxeles y densidad
Clasificación
Unión de cables con la misma pareja
Eliminación de cables otros sin contacto con mástiles
Guardar
Fin

12 LINKING

Entradas

- *Vx : voxels.*
  Nube de puntos voxelizada
- *Peaks : Nx1 numeric.*
  Índices de los vóxles en vx que se quieren unir
- *Step : numeric*
  Ancho de las secciones
- *widthElement: numeric*
  Ancho del tipo de elemento que se va a unir
- *keepingSeeds : numeric*
  Distancia máxima que se conserva una semilla sin propagación
- *Axis : N numeric.*
  Dimensiones que a tener en cuenta para la propagación

Salidas

- *Idx : Nx1 numeric*
  Vector que asigna a cada voxels peaks un índice, asignándolo a un grupo

Descripción

Esta función es útil para unir puntos pertenecientes a elementos lineales orientados. En este proceso se usa para unir los cables los cables, e inicialmente se usaba también para los raíles. Agrupa puntos ya segmentados mediante un algoritmo de RegionGrowing, creciendo todos los elementos de forma simultánea sección a sección. Para asignar un punto de una sección nueva a un grupo se compara dicho punto con los grupos anteriores, teniendo en cuenta la posición de la semilla de cada grupo, la media del grupo y el tamaño del grupo (si su longitud es mayor o menor que una sección), teniendo en cuenta únicamente las dimensiones especificadas en axis.

Primero, se secciona la nube en la dirección X, que debe ser la dirección de los elementos lineales, y se van recorriendo los *peaks* de cada sección por orden.

En la primera sección, si hay *peaks*, estos se agrupan por distancia. Pertenecen al mismo grupo si sus distancias son menores que *widthElement*/2. Se le asigna un número único a cada grupo y en la variable *idx*, en la columna de cada *peak*, se escribe su índice de grupo. En la variable *nextSeeds* se guarda el índice de un miembro de cada grupo, en *nextMeanCluster* su posición y *nextBigCluster* se ponen todos a falso. Un grupo tendrá a verdadero *nextBigCluster* cuando su rango sea mayor que step.

Analizada la primera sección, se ejecuta un bucle for desde la segunda hasta la última. Primero, se actualizan las variables sedes, *meanCluster* y *bigCluster*, a partir de las variables *nextSeeeds*, *nextMeanCluster* y *nexBogCluster*, eliminando los grupos cuyas semillas se encuentra más alejado que *keepingSeeds* de la sección.

Se actualiza la variable *nextPeaks*, que contiene los índices de los *peaks* en esta sección. Se inicializan las variables *nextSeeds*, *nextMenaCluster* y *nextBigCluster*. También se inicializa a falso la variable *visited*, que va registrando que que peaks en esta sección se analizan. Se inicializa a verdadero la variable *keep*, que se ira poniendo a falso a medida que las semillas seeds encuentren propagación en esta sección.

Inicilizadas las variables, se recorren todas las semillas en un bucle for. Primero, se buscan los *nextPeaks* que se encuentren a una distancia menor que *widthElement* en las dimensiones axis de la semilla a estudiar, guardando sus índices en *inRange*. La fila correspondiente a los puntos *inRange* en la variable *visited* se pone a verdadera, ya que estos *nextPeaks* están siendo analizados. Se calculan sus distancias a *meanCluster* del grupo al que pertenece la semilla, y se calcula cual es la que se encuentra a menor distancia. Analizadas las distancias se recorren todos los puntos *inRange* en un bucle for.

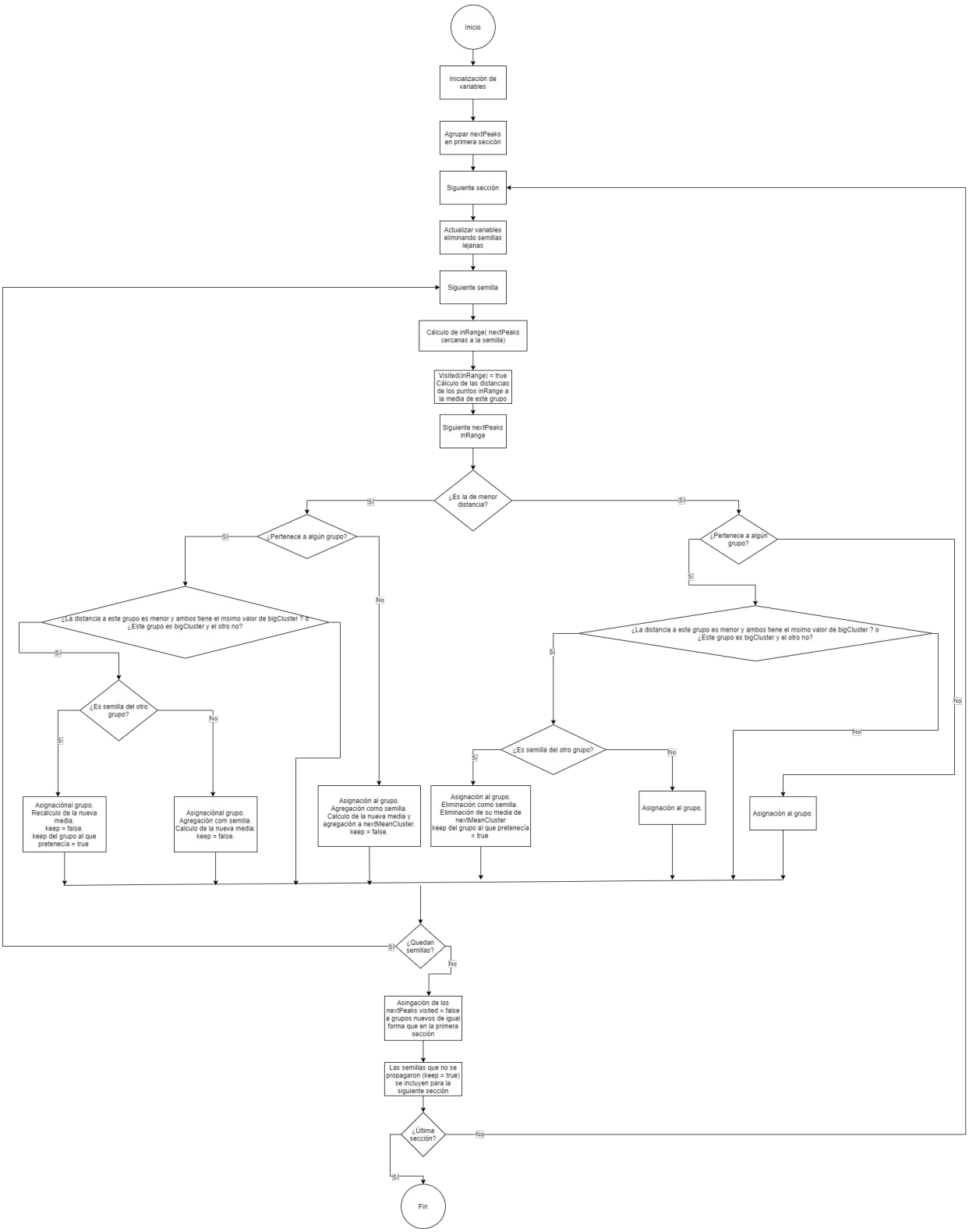
Si el punto es el que se encuentra a menor distancia y no pertenece a ningún grupo, su *idx* = 0, *keep* de esta semilla se pone a false, este *nextPeaks* se asigna al grupo de esta semilla, se añade este *nextPeaks* a *nextSeeds* y se añade la media de este grupo a *nextMeanCluster*, siendo la media entre la posición de este *nextSeeds* y el *meanCluster* del grupo. Este *nextPeaks* será la nueva semilla de este grupo en la siguiente sección.

Si este *nextSeeds* es el que se encuentra a menor distancia, pero pertenece a otro grupo, se cambia de grupo si ambos grupos tiene el mismo *bigCluster* y la distancia a la media de este grupo es menor o si este grupo es grande y el otro pequeño. Si se asigna a este grupo, pero era semilla del otro, se debe volver a verdadero la variable *keep* del otro grupo, ya que se quedó sin semilla.

Si no es la que se encuentra a menor distancia, se realizan las mismas comprobaciones para asignarla a un grupo, pero sin añadirla a *nextPeaks* ni acutalizando las variables *keep*, ni *nextSeeds* ni *nextMeanCluster*. Únicamente cuando se asigne al grupo y sea la semilla de otro, se actualizarán estas variables para eliminar lo relativo a este *nextPeaks*.

Después de analizar todas las semillas, se analizan los *nextPeaks* que no han sido visitados (*visited* = false). El procedimiento es igual que en la primera sección.

Por último, se añaden a *nextSeeds*, *nextMeanCluster* y *nextBigCluster* los grupos que no han tenido propagación (*keep* = true).


Inicio
Inicialización de variables
Agrupar nextPeaks en primera seción
Siguiente sección
Actualizar variables eliminando semillas lejanas
Siguiente semilla
Cálculo de inRange(nextPeaks cercanas a la semilla)
Visited(inRange) = true
Cálculo de las distancias de los puntos inRange a la media de este grupo
Siguiente nextPeaks inRange
¿Es la de menor distancia?
Si
Si
¿Pertenece a algún grupo?
Si
No
¿Pertenece a algún grupo?
Si
¿La distancia a este grupo es menor y ambos tiene el msimo valor de bigCluster ? o ¿Este grupo es bigCluster y el otro no?
Si
¿La distancia a este grupo es menor y ambos tiene el msimo valor de bigCluster ? o ¿Este grupo es bigCluster y el otro no?
Si
No
No
¿Es semilla del otro grupo?
Si
No
¿Es semilla del otro grupo?
Si
No
Asignaciónal grupo. Recálculo de la nueva media. keep = false. keep del grupo al que pretenecía = true
Asignaciónal grupo. Agregación com semilla. Calculo de la nueva media. keep = false.
Asignación al grupo. Agregación como semilla. Calculo de la nueva media y agregación a nextMeanCluster. keep = false.
Asignación al grupo. Eliminación como semilla. Eliminación de su media de nextMeanCluster keep del grupo al que pretenecía = true
Asignación al grupo.
Asignación al grupo.
¿Quedan semillas?
Si
No
Asingación de los nextPeaks visited = false a grupos nuevos de igual forma que en la primera sección
Las semillas que no se propagaron (keep = true) se incluyen para la siguiente sección
¿Última sección?
No
Si
Fin

13 DROPPERSEXTRACTION

Entradas

- *Vx : voxels*
  Nube de puntos voxelizada vxSec
- *Components : cell*
  Estructura componentsVxSec
- *dropperModel : Element*
  Modelo de dropper
- *distDbscan : numeric*
  distancia usada como argumneto de entrada en la agrupación de vóxeles

Salidas

- *components.droppers : cell 1 x número de components.cables.pairs*

Estructura. Cada estructura contiene las estructuras con los índices de los droppers pertenecientes a una pareja de cables.

- *Components.cables.others : cell 1x nº cables otros*
  Se añaden los cables de contacto y catenaria que no tienen droppers
- *Components.cables.pairs : cell 1x nº cables pareja*
  Se eliminan las parejas sin droppers

Descripción

El objetivo de esta función es la extracción de droppers. Los droppers son cables verticales que conectan los cables de contacto con los de catenaria. Al ser verticales no han sido extraídos en la función *cablesExtraction*.

Primero, se seleccionan todos los vóxeles que no han sido segmentados y se encuentren dentro del cuadro limitador que contiene todos los cables de contacto y catenaria de la nube. Los índices de estos vóxeles son *possibleDroppers*.

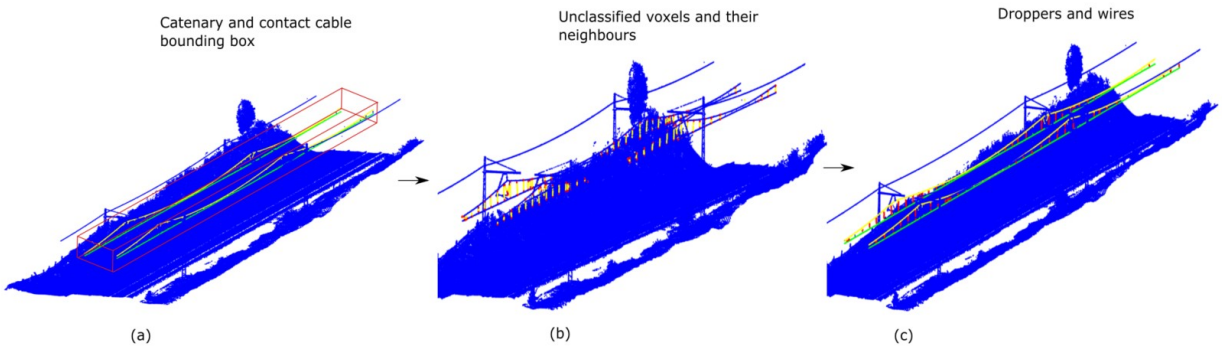
A *possibleDroppers* se le añaden sus vecinos con el fin de añadir los vóxeles de los cables en contacto con los droppers. Los índices de estos vóxeles son *droppers*.

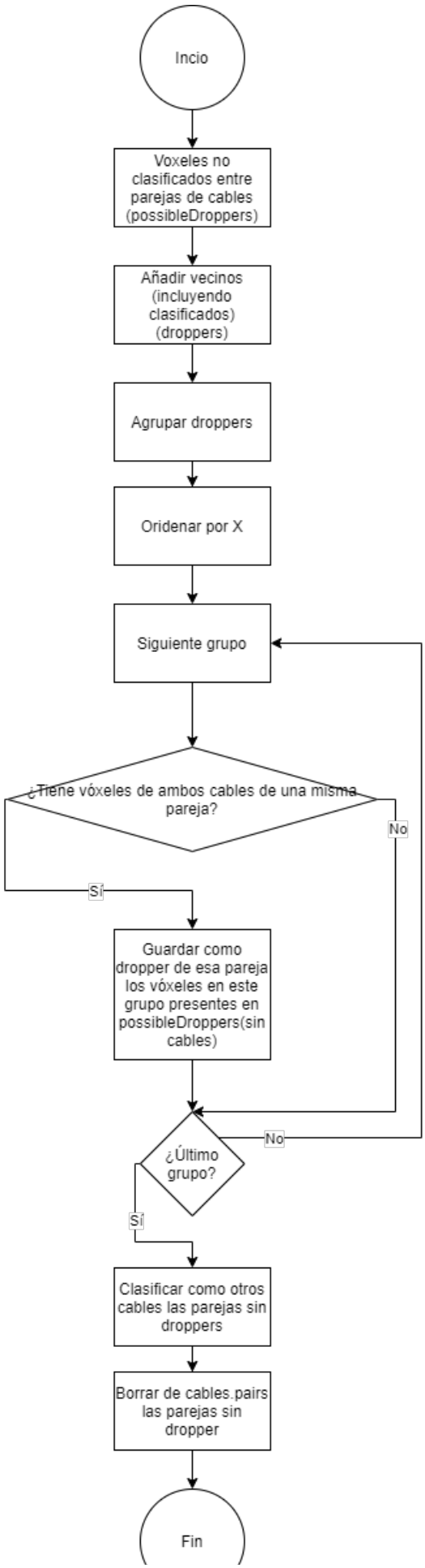
Los vóxeles de los índices *droppers* se agrupan y se ordenan por X. Ordenaros es útil para la función railsExtraction.

Cada grupo es analizado. Si tiene vóxeles presentes en el cable de contacto y catenaria de una pareja, se guardan sus índices en una celda dentro de la celda *components.droppers* correspondiente a esa pareja de cables. Solamente se guardan los índices de ese grupo presentes en *possibleDroppers* para no añadir índices que han sido clasificados como cables.

Las parejas de cables sin droppers son reclasificadas como otros cables.

En la función *railsExtraction*, los droppers encontrados en cables que no tienen raíles asociados son eliminados.


Catenary and contact cable bounding box
Unclassified voxels and their neighbours
Droppers and wires
(a)
(b)
(c)


Incio
Voxeles no clasificados entre parejas de cables (possibleDroppers)
Añadir vecinos (incluyendo clasificados) (droppers)
Agrupar droppers
Oridenar por X
Siguiente grupo
¿Tiene vóxeles de ambos cables de una misma pareja?
No
Sí
Guardar como dropper de esa pareja los vóxeles en este grupo presentes en possibleDroppers(sin cables)
¿Último grupo?
No
Sí
Clasificar como otros cables las parejas sin droppers
Borrar de cables_pairs las parejas sin dropper
Fin

14 RAILSEXTRACTION

Entradas

- *Vx : voxels.*
  Nube de puntos voxelizada vxSec
- *Components : cell*
  Estructura con los elementos extraídos
- *railPairModel : Element*
  Modelo de la pareja de raíles
- *marginSearch : numeric*
  Margen de búsqueda de los raíles desde la catenaria
- *heightFromGround : numeric*
  Distancia entre el suelo y el comienzo del raíl

Salidas

- *Components.rails : cell 1 x nº de components.cables.pairs*

Estructura que contiene parejas de raíles. Dentro de cada estructura, el {1} es el carril izquierdo y el {2} el derecho. Están ordenadas por el mismo orden que la pareja de cables a la que pertenecen. *Components.rails{1}* contiene los raíles izquierdo y derecho que se en encuentran bajo los cables *components.cables.pairs{1}.*

- *Components.cables.others : cell*
  Actualizado
- *Components.cables.pairs : cell*
  Actualizado
- *Components.droppers : cell*
  Actualizado
- *Components.masts : cell*
  Actualizado
- *Componetns.roughMasts : cell*
  Actualizado

Descripción

Esta función extrae los raíles buscándolos debajo de cada par de cables contacto-catenaria.

Primero, se rasteriza toda la nube, ya que se va a usara en cada iteración de búsqueda de raíles para cada pareja de cables.

De cada pareja de cables, se selecciona el más largo. Para extraer los raíles bajo este cable se debe delimitar qué parte de este puede contener raíles. Los cables empiezan y acaban en un mástil, pero no los raíles. Desde la última ménsula en contacto hasta el contacto con el mástil, el cable no tiene raíles bajo él. Por ello, se analiza el contacto del cable con los mástiles, las

ménsulas, y el número de droppers entre el principio/final del cable y la primera/última ménsula en contacto.

Para ello, primero se calcula el primer y último punto del cable. Después, se analizan los contactos, medidos por distancia, entre todos los mástiles y cualquier parte del cable, en su inicio y en su final. A mayores, se analiza el contacto con las ménsulas. Este último contacto se hace por vecindad, y se estudia únicamente el contacto con las ménsulas, descartando el mástil.

Una vez analizados estos contactos, se calcula el número de droppers antes de la primera ménsula en contacto y después de la última. Esto se puede hacer de una forma sencilla con un contador ya que se conoce la posición de las ménsulas y los dropper están ordenados en X.

Analizados todos los contactos, se escoge la sección del cable de la siguiente forma:

- Si el cable está en contacto con mástiles en puntos diferentes a su inicio o a su final, este cable no tiene raíles debajo de él.
- Si está en contacto con un mástil en su inicio o tiene menos de 2 droppers entre su inicio y la primera ménsula, y si está en contacto con un mástil en su final o tiene menos de 2 droppers entre su final y la última ménsula, solamente se estudia la zona entre la primera y la última ménsula. Si solamente tiene una ménsula en contacto, no hay raíles.
- Si está en contacto con un mástil en su inicio o tiene menos de 2 droppers entre su inicio y la primera ménsula, solamente se estudia la sección de cables posterior a la primera ménsula.
- Si está en contacto con un mástil en su final o tiene menos de 2 droppers entre su final y la última ménsula, solamente se estudia la sección de cables anterior a la última ménsula.
- Si no es ninguna de los casos anteriores se estudia todo el cable.

Definido el cable, se seleccionan sus píxeles en la imagen rasterizada. Por problemas de extracción puede que este cable no sea continuo. Estos problemas de discontinuidad en la imagen rasterizada se solucionan igualando las columnas en las que no hay ningún pixel a la columna anterior, dentro de los límites del cable.

Sobre la imagen rasterizada, se seleccionan los píxeles a la izquierda y derecha de los píxeles del cable. La distancia está definida por *railPairModel*, al cual se le aplica el margen *marginSearch*. Esta búsqueda se hace columna a columna (sentido de avance del cable). Es necesario comprobar que los límites de búsqueda se encuentren dentro de los límites del ancho de las filas de la imagen rasterizada. Como resultado se obtienen los vóxeles contenidos en los píxeles a la izquierda y a la derecha de la sección de cable.

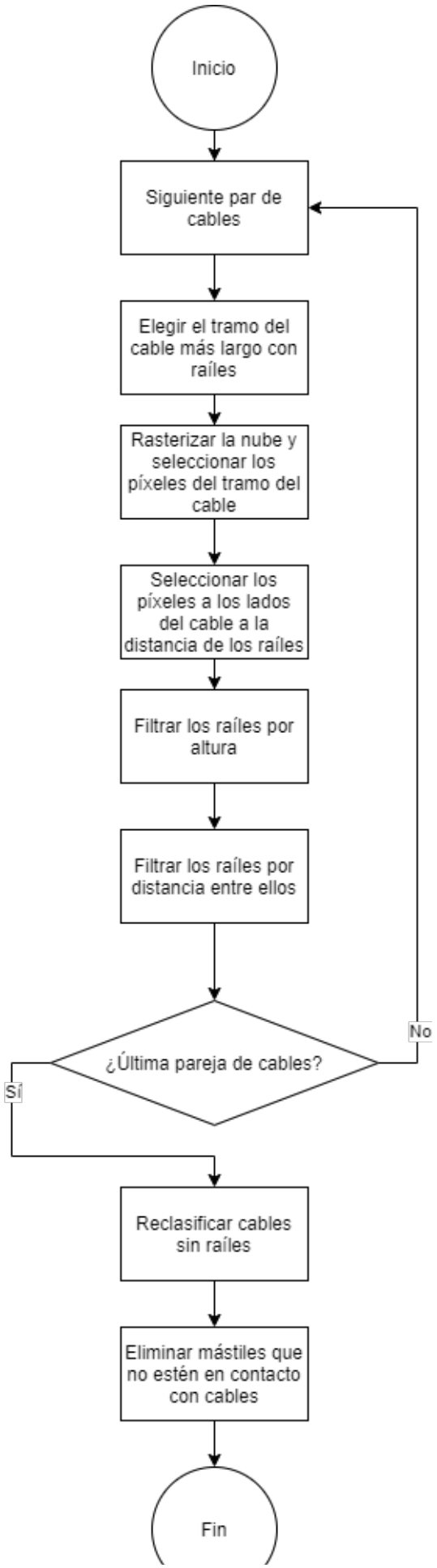
Una vez seleccionando los vóxeles de los raíles, estos se filtran por separado. Para ello, se estudia cada sección de raíl. En cada sección se calcula el histograma de alturas. La altura con mayor frecuencia es la del suelo, por lo que se seleccionan los vóxeles cuya altura sea la del suelo más *heightFromGround*. Para poder hacer esto, el raíl es orientado usando *PcaFlattering* y centrado en su centro de masas.

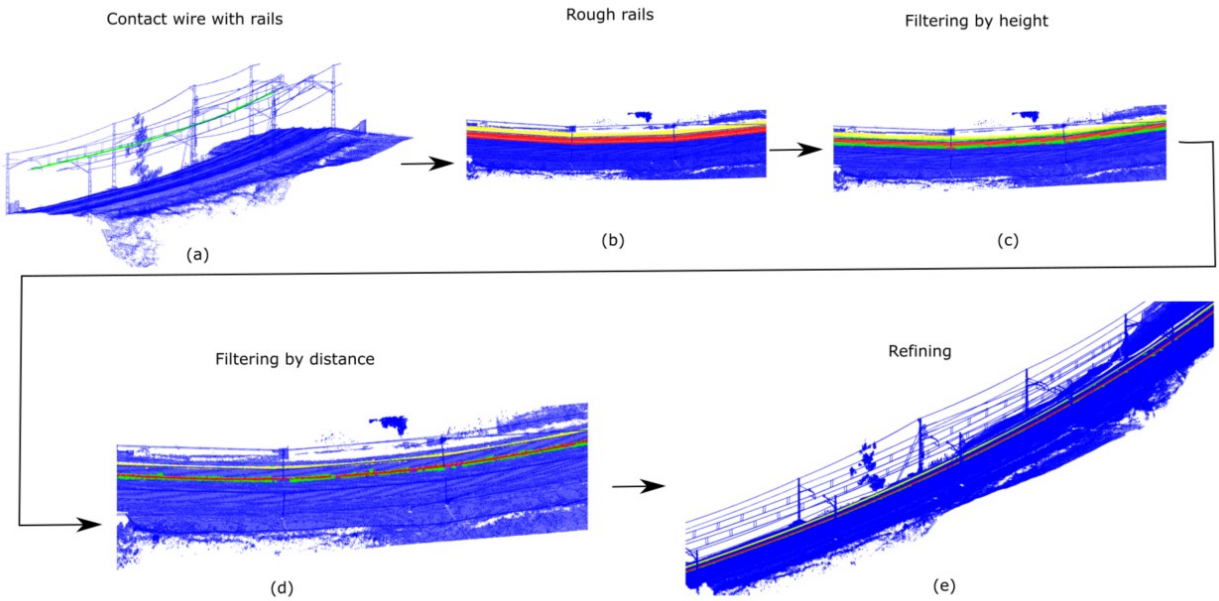
Una vez filtrados los raíles de forma independiente, se filtran por distancia entre ellos, usando la distancia definida por *railPairModel*. El proceso se hace sobre la imagen rasterizada, siendo similar al de la selección de píxeles a partir del cable.

Los índices de cada raíl se guardan en una estructura, siguiendo el mismo orden que las parejas de cables.

Segmentados los raíles, las parejas de cables que no tienen raíles son reclasificadas como otros.

Por último, se comprueba que todos los mástiles tengan al menos un cable clasificado como otro en contacto, de lo contrario son eliminados.


Inicio
Siguiente par de cables
Elegir el tramo del cable más largo con raíles
Rasterizar la nube y seleccionar los píxeles del tramo del cable
Seleccionar los píxeles a los lados del cable a la distancia de los raíles
Filtrar los raíles por altura
Filtrar los raíles por distancia entre ellos
¿Última pareja de cables?
No
Sí
Reclasificar cables sin raíles
Eliminar mástiles que no estén en contacto con cables
Fin


Contact wire with rails
Rough rails
Filtering by height
(a)
(b)
(c)
Filtering by distance
Refining
(d)
(e)

15 SIGNALSEXTRACTION

Entradas

- *Vx : voxels*
  Nube de puntos voxelizada vxSec
- *Components : cell*
  Estructura con los elementos segmentados
- *percentHighestInt : numeric*
  Porcentaje mínimo de puntos de alta intensidad en un mástil con señal
- *inMast : numeric*
  Distancia máxima de los puntos considerados al centro del mástil en XY.
- *Model : array of Element*
  Vector con los modelos de las señales
- *Neighborhood : numeric*
  Nivel de vecindad considerada para la recuperación de palos segmentados como suelo.

Salidas

- *Componenents.signals.big : cell 1x nº de señales*
  Estructura con los índices de los vóxeles de cada señal
- *Components.signals.trafficLight : cell 1 x nº señales de tráfico*
  Estructura con los índices de los vóxeles de cada señal de tráfico
- *Components.signals.stone : cell 1x nº de marcadores*
  Estructura con los índices de los vóxeles de cada marcador

Descripción

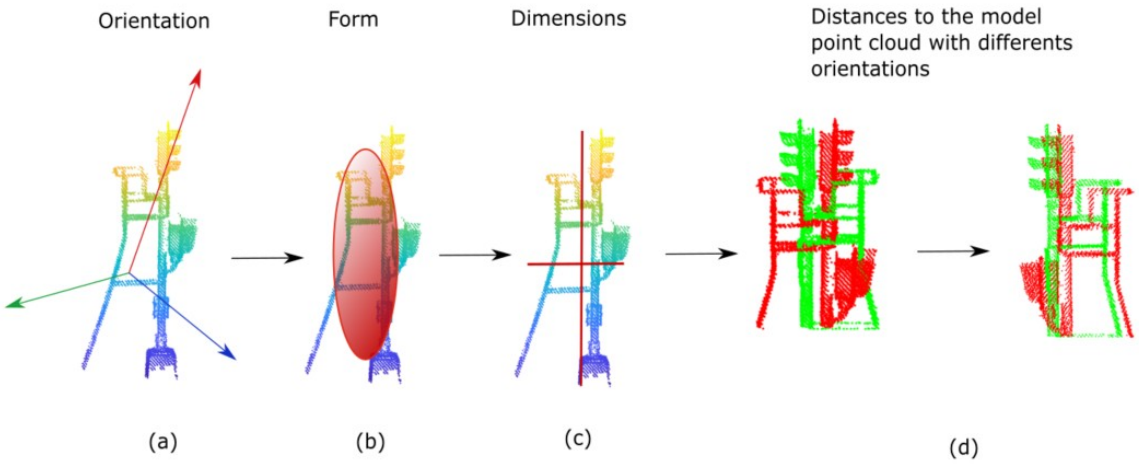
Esta función distingue 2 partes: la extracción de señales en mástiles y la extracción de señales aisladas.

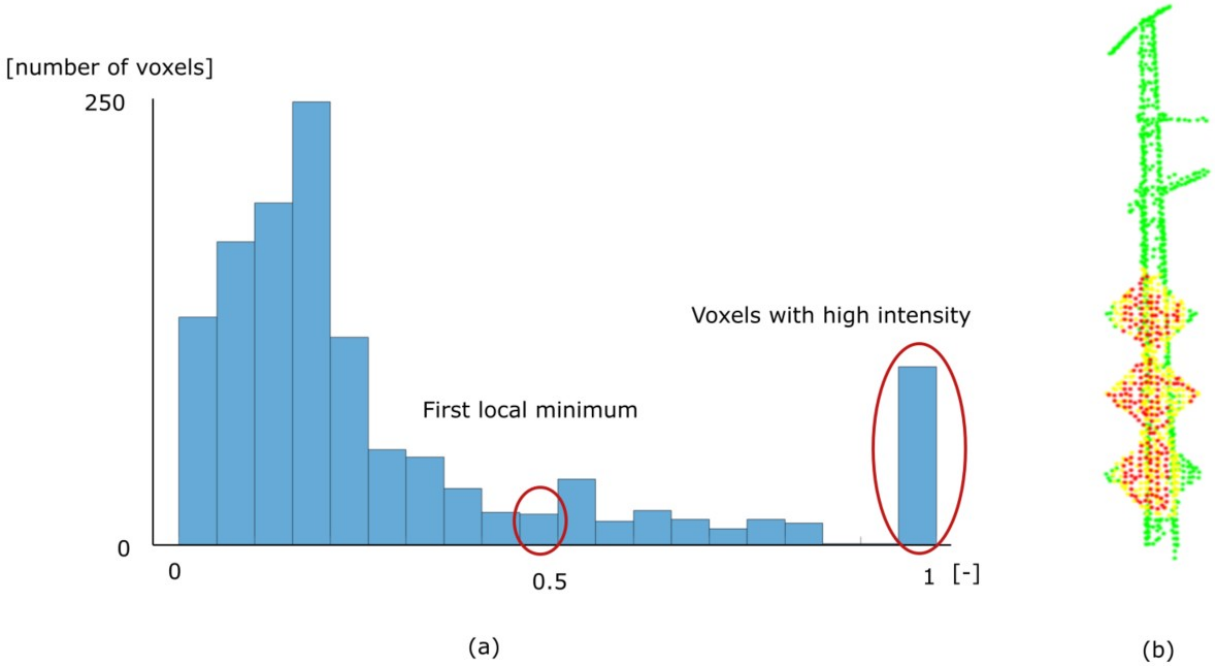
La extracción de señales en mástiles está basada en la gran intensidad de estas. Para su extracción, se analizan los puntos de cada mástil. Por si el mástil no estuviese completo, se calcula su centro en las dimensiones X e Y y se seleccionan todos los vóxeles en notTrack a una distancia menor que inMast. Seguidamente, las intensidades de los puntos seleccionados se normalizan y se calcula un histograma. Si el porcentaje de puntos en la barra de más intensidad es superior a percentHightstInt se procede a buscar señales en dicho mástil. Para ello se busca el primer mínimo local en el histograma, y se seleccionan y agrupan todos los vóxeles con una intensidad mayor a dicho mínimo. Se descartan los grupos con pocos vóxeles, y a los grupos seleccionados se le añaden sus vecinos. Estos vóxeles se segmentan como señal en mástil.

Para la extracción de señales aisladas se seleccionan todos los vóxeles en notTrack que no hayan sido segmentados. Acto seguido, se le añaden los vecinos justo debajo de estos vóxeles, tantas veces como neighborhood. Esto se hace para recuperar los vóxeles que por error se han segmentado como track y pertenecen a un mástil. Se asume que el error puede ser de 1 voxel en

la nube de extracción del suelo, por lo que neghborhood es tamaño voxel extracción suelo / tamaño voxels vx. Los vóxeles seleccionado se agrupan y se descartan aquellos con pocos vóxeles.

Cada grupo, si no tiene ningún voxels perteneciendo a wall2 es analizado usando la función CheckElement, comparando cada grupo con su elemento Element. Para tener más resolución se usa los puntos de la nube padre de vx. El objeto se orienta con la trayectoria, y se centra en su centro de masas en los ejes XY y se enrasa por su lugar más alto. Esto permite que el grueso de las señales, su lugar más alto, estén siempre centrados en el mismo punto, independientemente de la longitud de su mástil, que puede ser variable o estar mal segmentado.





16 LINKINGBETWEENSECTIONS

Entradas

- *vxLocation : Nx3*
  Coordenadas XYZ de la nube de puntos *vx*
- *elementsPre : cell.*
  Estructura con los índices de los elementos en la sección anterior. Si son elementos independientes elementsPre{i} es el elemento i. Si los elementos están agrupados, como en el caso de los cables contacto-catenaria, elelentsPre{i}{2} es el cable de catenaria de la pareja i.
- *elementsSec : cell*
  Estructura con los índices de los elementos en esta sección
- *trajPoints : Nx3*
  Puntos de la trayectoria de la sección
- *Clusters : numeric nº grupos sección pre.*
  Vector que identifica el grupo de los elementos en la sección anterior en *componentsVx*
- *maxDistX : numeric*
  Distancia máxima en X entre elementos en distintas secciones para ser clasificados como el mismo elemento.
- *maxDistY : numeric*
  Distancia máxima en Y entre elementos en distancias secciones para ser clasificados como el mismo elemento.

Salidas

- *newClusters* : numeric nº grupos en esta sección.
  Vector que identifica el grupo de los elementos en esta sección en *componentsVx*

Descripción

Función que analizar elementos continuos de la misma clase en secciones consecutivas para determinar a qué elemento de la sección anterior pertenecen o si forman un elemento nuevo. Se usa para parejas de cables y cables clasificados como otro. Los cables otros se analizan independientemente de los cables contacto-catenaria. Los cables de contacto se analizan por separado de los de catenaria inicialmente, pero ambos son asignados a la misma pareja, escogiendo la pareja que muestre una mejor continuidad, ya sea contacto-contacto o en su catenaria-catenaria.

Primero, se corrige *elementsSec* y *elementsPre* para que tengan la misma estructura sean o no elementos independientes.

Después, en *elements* se guardan los elementos en la sección anterior y en esta, por orden, guardando en *idx* el índice que identifica a que grupo pertenecen. *Elements{1}* contiene todos los índices de los elementos de tipo 1 (cables de contacto por ejemplo), *elemnts{2}* los del 2, etc.

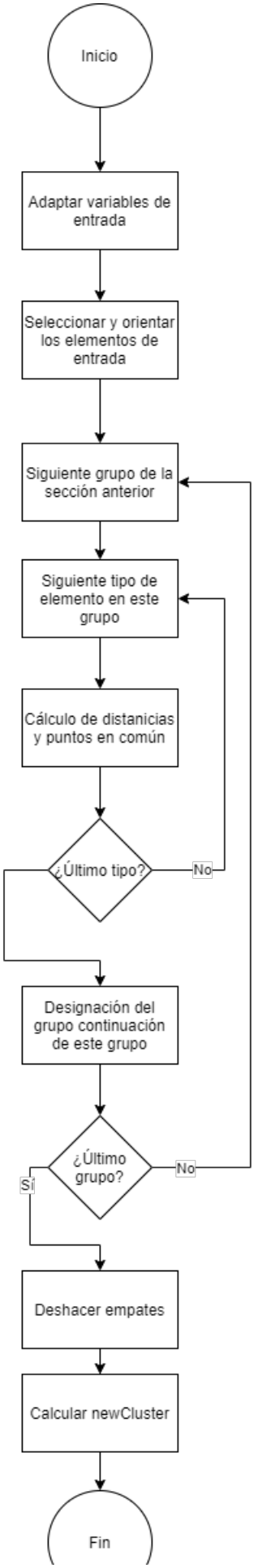
Los elementos se seleccionan de la nube y se orientan usando *trajPoints*, que es la trayectoria de la sección.

Se inicializan las variables *nextCluster* y *distances*. *nextCluster* es un vector que determina qué elemento de esta sección es la continuación del elemento de la sección anterior. Por ejemplo, si nextCluster(4) = 3, quiere decir que el elementsSec{3} es la continuación de elementsPre{4}. La variable *distances* contiene la distancia mínima entre los elementos en la sección anterior y en su continuación, medida en el eje Y. Es decir, si *distances*(3) = 1'2, esto indica que la distancia entre el *elementsPre{3}* y su continuación es de 1'2 en Y. Si las catenarias se encontraban a 1'2 y los cables de contacto a 1'4, la que define su distancia es 1'2 por ser la menor.

Se analizan todos los elementos en *elementsPre*. Para cada elemento, se calculan las distancias en X y en Y y los puntos en común, en caso de haber solapamiento, entre el elemento en la sección anterior seleccionada y todos los elementos en esta sección, únicamente con los que son de la misma clase (contacto con contacto, catenaria con catenaria). Las distancias se miden entre el final del elemento en la sección anterior y el primer punto de los elementos en esta sección cuya X sea mayor que a X del final del elemento en la sección anterior que se está estudiando. Analizadas todas las distancias, se descartan las que no son menores que *maxDistY* y *maxDistX* por estar demasiado lejos. De los elementos en esta sección que cumplan las especificaciones, se escoge como continuación aquel que tenga más puntos en común entre elementos del mismo tipo (contacto-contacto o catenaria-catenaria). De haber empate, se escoge el que tenga menor distancia en Y entre los elementos del mismo tipo. Si no se encuentra continuación para el elemento j *nextCluster*{j} = 0.

Debido a que se compara con todos los elementos podría asignarse el mismo elemento en esta sección a varios de la sección anterior. Para corregir esto, se mantiene el que esté a menos distancia según *distances*.

Por último, se calcula la variable *newClusters*. Esta variable representa lo mismo que *clusters* pero para los elementos de esta sección en vez de para los de la sección anterior. La variable *clusters* contiene el grupo al que pertenece cada *elementsPre* en *componentsVx*. Si *clusters*(1) = 5, quiere decir que el *elementsPre*{1} es el grupo 5. Si *nextCluster*(1) = 3, esto indica que *elementsSec*{3} es la continuación de *elementsPre*{1}. Por consiguiente, *newClusters*(3) = 5.


Inicio
Adaptar variables de entrada
Seleccionar y orientar los elementos de entrada
Siguiente grupo de la sección anterior
Siguiente tipo de elemento en este grupo
Cálculo de distanicias y puntos en común
¿Último tipo?
No
Designación del grupo continuación de este grupo
¿Último grupo?
No
Sí
Deshacer empates
Calcular newCluster
Fin

17 MERGINGRAILS

Entradas

- *vxLocation :Nx3 numeric*
  .Location de la nube
- *traj : Trajectory*
  Objeto de la clase *Trajectory*
- *Rail : cell*
  Estructura con los índices de los raíles organizados por parejas.

Salidas

- *newClusters: Nx1 numeric*
  Índices con el grupo de raíles al que pertenece cada pareja en *Rail*

Descripción

El objetivo de esta función es unir las parejas de raíles que son en realidad el mismo raíl y fueron separados al ser dependientes de cables de contacto y catenaria diferentes. Los raíles en la nube son orientados con la trayectoria. En cada pareja de raíles, se determina cual es la pareja de raíles que son la continuación de este. Para ello se analizan los vóxeles en común y se determina si dichos vóxeles son el final de la pareja estudiada y el principio de la siguiente. De no ser así podría tratarse de un cruce de raíles.

Determinados cuáles son sus continuaciones, se les asigna el mismo índice en la variable *newClusters* a todas las parejas que pertenecen al mismo raíl.

18 DENOISINGRAILS

Entradas

- *Vx : voxels*
  Nube de puntos voxelizada *vx*
- *railIdx : Nx1 numeric*
  Índices del raíl en la nube padre
- *Step : numeric*
  Ancho de las secciones del raíl para su filtrado
- *railModel : Element*
  Modelo del raíl

Salidas

- *Rail : Nx1 numeric*
  Índices del raíl filtrado en la nube padre de *vx*

Descripción

Esta función se aplica para filtrar cada raíl, eliminando los puntos pertenecientes al balasto. Este filtrado se realiza sobre la nube de puntos padre de *vx*, una nube sin voxelizar, ya que debido al gran tamaño del voxel no es posible realizar este filtrado en *vx*.

Primero, se seleccionan los puntos en la nube de puntos padre pertenecientes al raíl, y se centran y se orientan.

Después, el raíl se secciona y se analiza cada sección por separado. En cada sección se hace un histograma del eje Y buscando la coordenada con mayor frecuencia, que es la posición central del raíl. Sobre esa coordenada se seleccionan los puntos a la izquierda y a la derecha dentro del rango especificado por *railModel*. Se calcula la media en Z de los puntos seleccionados y se seleccionan los que tengan una Z superior a la media.

Antes de ejecutar esta función, se analizaron los vóxeles compartidos entre raíles. Se calcula con cuantas parejas de raíles una pareja comparte algún punto, y se introducen en esta función de parejas con menor contacto con otras a más. A mayores, los puntos asignados a un raíl se eliminan de los siguientes grupos. De esta forma los raíles que van de una vía a otra (raíles para un cambio de vía), son los últimos a los que se le aplica la función DenoisingRails, y únicamente estarán formados por puntos que no pertenezcan a ningún otro raíl.


Inicio
Centrado y orientación del raíl en la nube sin voxelizar
Siguiente sección del raíl
Filtrado por histograma en Y
Filtrado en Z por media
¿Últimа sección?
No
Sí
Fin

19 MODIFYSAVELAS

Entradas

- *PathIn: char*
  Localización del archivo de la nube
- *PathOut: char*
  Carpeta en la que se guarda la nube modificada
- *components : cell*
  Estructura resultado de la segmentación

Salidas

Se carga la nube, se escribe en ella y se guarda en *PathOut* con el mismo nombre que la nube de entrada. Si no se especifica, se sobrescribe. Para leer y escribir en el .las se utiliza la librería LASio.

Descripción

La salida es la nube .las de entrada en la que se modifican los siguientes campos.

- Record.classification: asigna un número a cada punto en función del tipo de elemento al que pertenezca
  - Rail = 1
  - Catenaria = 2
  - Contacto = 3
  - Dropper = 4
  - Otros cables = 5
  - Mástiles = 6
  - Señales = 7
  - Semáforors = 8
  - Marcadores = 9
  - Señales en mástiles = 10
  - Luces = 11
- Record.user_data: asigna un número a los elementos que pertenecen a una misma vía. Estos elementos son raíles, cables de contacto, catenaria y droppers. Todos los elementos de una misma vía tienen un número comprendido entre una potencia de 10. A los raíles se le asigna una potencia de 10 ($i \cdot 10^x$). Al grupo de cables j de contacto y catenaria con sus droppers pertenecientes a la vía i se les asigna el valor $i \cdot 10^x + j$. El número máximo de cables de contacto-catenaria de cada vía no puede ser superior a 9.
- Record.point_source_id: asigna un número a cada punto en función del grupo al que pertenezcan. Cada grupo tiene un número único.

Si no es capaz de segmentar una nube, guarda una variable con un mensaje de error en la ruta especificada para ello. No se analiza el tipo de error.

Las nubes deben tener georreferencia 3D, intensidad y sellos de tiempo.

La trayectoria debe tener georreferencia 3D y sellos de tiempo sincronizados con los de las nubes.

20 CLUSTERINGNEIGHBORS

Entradas

- *Vx : Voxels*
  Nube voxelizada
- *Vargin{2} : char*
  Modo
- *Varargin{3}*
  Variable necesaria en función del modo

Salidas

- *clusterIndex*
  Vector que especifica a qué grupo pertenece cada voxel

Descripción

Función para agrupar los vóxeles en vx mediante un algoritmo de Regiongrowing usando la vecindad. Los modos son:

- Modo predeterminado. Se selecciona una semilla y se asignan a ese grupo todos los vóxeles que sean vecinos entre ellos
- Distance. Igual que el anterior, pero a mayores deben de esta a una distancia menor que la especificada.
- El resto de modos no se usan en el código.

21 SAVEPARALLEL

Función para escribir en disco en un bucle paralelo. Se usa porque Matlab no deja usar la función save() dentro de un bucle paralelo, pero sí si se llama desde una función dentro de un bucle paralelo.

22 ELEMENT (CLASS)

Clase diseñada para generar modelos de elementos y compararlos con una nube de puntos mediante el método *CheckElement*.

Atributos

- *Dimensions*: 1x3 numeric.

  Dimensiones del objeto en XYZ en las mismas unidades que la nube de puntos de *points* y en la nube de entrada de *CheckElement*
- *Eigenvectors*: 3x3 numeric

  Matriz con los autovectores del PCA del objeto ordenados por columnas de mayor a menor autovector.
- *Eigenvalues* : 1x3 numeric

  Vector con los autovalores del PCA en tanto por uno del objeto ordenados de mayor a menor.
- *Points* : Nx3 numeric

  Nube de puntos modelo del objeto. En este caso orientada con la trayectoria y centrada en su centro de masas y enrasada en Z con su punto más elevado. En este algoritmo no se usa el palo de las señales porque su longitud es variable.
- *toleranceDimensions* : 1x3 numeric.

  Tolerancia admisible en las dimensiones en las mismas unidades que *dimensions*
- *toleranceEigenvectors* : 1x3 numeric

  Tolerancia en grados de la dirección de cada *autovector*.
- *toleranceEigenvalue* : 1x3 numeric

  Tolerancia de los autovalores
- *tolerancePoints* : numeric.

  Máximo admisible de la media de las distancias al cuadrado entre la nube *points* y la nube de entrada de *CheckElement*.

Los atributos no introducidos por defecto son NaN, no teniendo en cuenta esas especificaciones.

CheckElement

Entradas

- *points* : Nx3 numeric

  Nube de puntos del elemento a estudiar
- *Element* : Element

  Modelo con el que se va a comparar

Salidas

isElement : logical
Indica si *points* es un elemento como *element* o si no

Descripción

Método para comparar una nube de puntos de un objeto con un modelo. La nube de puntos debe estar orientada de igual forma que el modelo. En este algoritmo se orientan los objetos con su trayectoria, y se centran en su centro de masa en XY y se enrasan su parte más elevada en Z con el origen. Para ser considerado un objeto igual que el del modelo debe de cumplir con todas las especificaciones del modelo (las que no son NaN).

En el estudio de las dimensiones, si no points no cumple las especificaciones con su orientación inicial, se orienta aplicando PCA a points. Esto se hace debido a que puede haber objetos con cierta inclinación, como el caso de señales. Tras hacer PCA, se reordenan sus ejes de modo que se trate la mayor componente de cada autovector como el nuevo eje. Por ejemplo, si es una señal vertical inclinada, el primer autovector será vertical con cierta inclinación, por lo que la dispersión en ese eje no es el nuevo X, es el nuevo Z.

En el cálculo de distancias entre el modelo y el objeto se usa los puntos únicamente con una Z superior a la mínima del modelo. De esta forma se elimina los palos de las señales.

Para hacer el cálculo de distancias, se calcula los puntos más cercanos en el modelo al objeto a estudiar y se computa la media de sus distancias al cuadrado. Del mismo modo, se hace la operación inversa, las distancias a los puntos más cercanos en el objeto al modelo. Es la mayor de ellas la que debe ser menor que la tolerancia especificada. De no ser así, se rota en Z el objeto, ya que puede que la señal esté apuntando en el otro sentido de la trayectoria. De no cumplirse, puede ser debido a una cierta inclinación, por lo que ser orienta con su PCA tanto el objeto como el modelo, centrando y se enrasando ambos por su punto más elevado. De seguir sin cumplir con la tolerancia, se rota en su eje X por el mismo motivo por el que antes se rotó en Z (ahora X por estar orientado y ser las señales elementos verticales).

23 POINTCLOUD_ (CLASS)

Modificación de la clase pointCloud de Matlab para añadirle el atributo intensidad.

24 RASTER2D (CLASS)

Clase heredada del código *Cloud2Aligment* para la rasterización de una nube de puntos desde vista de pájaro.

Se eliminó el reajuste del grid para poder medir distancias en píxeles. Al variar eso, fue necesario varia el round() en la asignación de pixel por ceil().

25 TRAJECTORY (CLASS)

Clase heredada del código *Cloud2Aligment* para la generación de un objeto de trayectoria a partir de un archivo en ASCII.

26 VOXELS (CLASS)

Clase que voxeliza una nube de puntos *pointCloud* o *pointCloud_*.