|  |  |  |  |
| --- | --- | --- | --- |
| Module | Parameter | Lokale Variablen | Funktion |
| FindMinMax | Pointlist | Imin, imax, jmin, jmax, iSplits, jSplits, iDistance, jDistance | * Die minimas und maximas der i und j-Werte der Koordinaten werden gesucht, um den „Rahmen“ des Gitters um das Schachbrett festzulegen * In den ConstantArrays JSplits und ISplits werden die Zellen des Gitters gespeichert. Diese werden über die Distanz der jeweilien Minimalwerte und Maximalwerte geteilt durch die gewünschte Anzahl an Zellen geteilt. |
| SortPointList | iSplits, jSplits, Pointlist | pi,pj | * Die Eckpunkte werden zunächst der Größe nach nach ihren i-Werten Sortiert. Die sortierte Liste wird dann durchgezählt, so dass jeder Punkt seinen Indexwert in I-Richtung bekommt * Danach werden die Eckpunkte der Größe nach nach ihren J-Werten sortiert und bekommen hier ebenfalls einen Index zugeordnet * (Diese Sortierung ist nach jetzigem Stand des Algorithmus vllt nicht mehr zwingend notwendig) |
| GoThroughConvexHulls | iSplits, jSplits, pj | ConvexHull | * Nun wird herausgefiltert, welcher Punkt in welche Zelle des erstelllten Gitters gehört, somit wird eine grobe Vorsortierung der Punkte für den weiteren Verlauf vorgenommen. * In einer For-Schleife welche alle iSplits durchzählt wird die Funktion FindPointsInConvexHull bei jedem Durchgang aufgerufen welche eine Liste mit Associations in die Liste ConvexHull hinzufügt. * Der Funktion werden die momentanen iSplits der Durchzählung übergeben und alle Jsplits. Des Weiteren wird die nach J sortierte Punkteliste übergeben |
| FindPointsInConvexHull | iSplits[[1,ii]], iSplits[[1,ii+1]], jSplits, pj | ConvexHullCell={}, ConvexHullList,ConvexHullCellKeys = <||> | * Eine Liste namens ConvexHullCell und eine Association nachmes ConvexHullKeys wird angelegt * Zwei For-Schleifen werden gestartet. Die erste läuft durch alle Jsplits, die zweite geht alle Punkte von pj durch. * Innerhalb der For-Schleife wird dann überprüft, welche Punkte aus pj sich innerhalb der übergebenen Isplits und den dazugehörigen jSplitts befinden. * Die Koordinaten, die Indizes und die Zellenbezeichnung werden dann in Keys in die Association ConvexHullCellKeys gespeichert und er Liste ConvexHullCell angehängt. Diese Liste wird dann and die Liste ConvexHull angehängt   Wiederholung des Vorganges mit neuen iSplits. |
| FindStartVectors | ConvexHull | StartPointCloud={}, StartPointCloudKeys=<||>, VecI,VecJ,countI,countJ,nectI,nextJ, | * DiePunkte der Zellen (1,1), (1,2) und (2,1) werden in eine neue Liste namens StartPointCloud gespeichert. * Die Liste wird durchlaufen, jedoch nur die Punkte deren Zellenkeys (1,1) aufweisen, dabei wird der geringste Wert von i und j aus ihnen ermittelt. * Sind beide Werte dem selben Punkt zugeordnet so ist dieser Punkt der StartPunkt für die Gittererzeugung des Schachbretts. (Kissen vezeichnungen, Rotationen) * Bei Tonnenverzeichneten nicht rotierten Bildern, kann es sein das die minimalwerte nicht einem Punkt entsprechen, dann wird der geringste j mit dem nächst geringstem i genommen. * Ist der StartPunkt ermitteln werden die nächsten Punkte in J- und I-Richtung gesucht. * Hierzu wird eine For-Schleife durchlaufen, welche den nächst kleineren Punkt in I- und J-Richtung vom Start aus hat. * Da das noch nicht sicher stellt, das das auch der gewollte Punkt ist, wird noch eine Prüfung und gegebenenfalls Austausch des potentiellen Punktes durchgeführt. * Es wird überprüft der nächste potentielle Punkt in J-Richtung dahingehend überprüft, ob die Distanz zwischen Start dem neuen Punkt in I-Richtung die minimalste ist, ist dies nicht der Fall werden die anderen Punkte dahingehend untersucht und der Punkt mit dem geringsten Distanz in I-Richtung genommen. * Selbiges wird auch für die J-Richtung vollzogen * Somit kann man mit den entstehenden Richtungsvektoren in beide Richtungen die neuen Punkte ermitteln. |
|  | nextI, nextJ, Start, ConvexHull | IList={}, JList={}, IDir, JDir, distance, cache, PotNextI, PotNextJ | * IDir und JDir sind die Richtungsvekoren vom Startpunkt aus in beide Kantenrichtungen des Schachbretts. * Danach werden die ersten beiden Spalten in I- und J-Richtung jeweils durchlaufen, und in iList und JList gespeichert. * Diese Listen enthalten alle potentiellen Punkte entlang der gesuchten Kante. * ... |
|  | IList, JList ,Start, nextI, nextJ, ConvexHull | SortedPointsKeys = <||>, Sortedpoints = {} | * StartPoint und NextPointI und NextPointJ werden die Keys NeighbourI und NeighbourJ gegeben mit StartPoint(NeighbourJ 🡪 1, NeighbourI 🡪 1), NextPointI(NeighbourJ 🡪 1, NeighbourI 🡪 2) und NextPointJ(NeighbourJ 🡪 2, NeighbourI 🡪 1). * Diese drei bereits bekannten Punkte werden auch in eine CheckPointListe gespeichert, diese wird für das spätere Prüfen von weiteren Punkten benötigt. * Nun wird zunächst in einer For-Schleife die Punkte von StartPoint und NextPointJ aus gesucht. * Für die Punkte in I-Richtung des Schachbretts wird das selbe Verfahren angewandt. * Benötigt wird die Distanz zwischen einem dem Momentanen StartPointJ, welcher mit jedem Durchlauf der Schleife mit neu gefundenem Punkt weiter gegeben wird und einem momentanen NextPointJ, welcher ebenfalls mit jedem Schleifendurchlauf weiterwandert, und die Richtung DirJ zwischen diesen beiden Punkten. * Die Schleife selbst durchläuft alle Punkte, welche in der für die Richtung entsprechenden Richtung Liste sind. In diesem Fall die JList * Innerhalb dieser Liste wird derjenige Punkt gesucht welcher zum NextPointJ den geringesten Abstand in J Richtung hat und dessen Abstand in I-Richtung einen gewissen proportion-Wert, welcher über ein Verhältnis ausgerechnet wird. Nicht über bzw unterschreitet, da der Punkt innerhalb des J-Abstandes in positiver oder negativer Richtung des NextPointJ sich befinden kann. Der Abstand in J-Richtung darf gleichzeitig auch nicht geringer als die vorgegebene Distanz des StartPointsJ zu NextPointJ sein. 🡪 dies schließt aus, dass bei einem rotierten und perspektivischen Schachbrett (Vgl TestObjekt4) ein Punkt in der falschen Zeile als Potentieller nächster Punkt genommen wird. * Ist der nächste Punkt gefunden, so wird dieser der SortedPointsList und der der CheckPointsList übergeben mit den passenendn Neighbour associations. * Des Weiteren bekommt für den nächsten Schleifendurchlauf StartPointJ die Werte von NextPointJ und NextPointJ wird der neu gefundenen Punkt aus der JListe gegeben. * Im Anschluss werden noch in AppendTo[SortedPoints,SaftyListJ[Start,CheckPointJ,proportionY,CheckCellForJ,ConvexHull,distanceJ]];   AppendTo[SortedPoints,CompleteJGrid[ nextI,ConvexHull,StartDistanceJ,StartProportionJ,Start ,Jp,aI]]  Weitere Punkte zur SortedList in J-Richtung hinzugefügt, bei ersterem nur in bestimmten Fällen. Mehr zu den Funktionen folgt.   * Nicht zu vergessen: selbiges wie Oben wird auch mit den Punkten in I-Richtung vollzogen, bis auf die CompleteGrid Funktion |
| SaftyList | Start, CheckPointJ, proportionY, CheckCellForJ, ConvexHull,, distanceJ | SaftyList = {}, SaftsKeys = <||>, SaftyKeysList = {} | * Der Funktion werden die Parameter CheckPointJ und CkeckCellForJ mitgegeben. Diese stammen aus der Funktion FindNeighbours und es handelt sich um den letzten Punkt der innerhalb der JListe ermittelt wurde und dessen i-Zelle in welcher sich dieser befindet. * Da die I- bzw die JListe in jede Richtung nur die beiden ersten Zellen untersucht werden, kann es bei einem rotierten Schachbrett sein, dass sich noch weitere Punkte in Zellen weiter oben/unten befinden * Die Funktion SaftyList, erstellt eine Liste aus möglichen weiteren Punkten, indem sie die in diesem Falle I-Zelle des letzten Punktes nimmt und diese so wie die unter und oberhalb dieser Zelle und alle deren J-Zellen aufwärts auf einen möglichen nächsten Punkt untersucht. * Sollte es noch einen geben wird dieser der CheckPointList und der SortedPointsList zugewiesen, ansonsten passiert nichts. * Das Auswahlverfahren läuft genau so ab wie in FindNeighbours. |
| CompleteJGrid | StartPoint, ConvexHull, StartDistanceJ, proportion, Start, J, aI | PossiblePointsList = {}, SortedPointsKeys = <||>, SaftyPossiblePointsListJ = {}, | * Nachdem der “Rahmen” des Schachbretts und die äußersten Punlte gefunden wurden, muss nun das restliche Grid des Schachbretts dedectiert und den richtigen Nachbarn zugeordnet werden. * Jeder Punkt der in I-Richtung als „Rahmenpunkt“ dedektiert wurde, wird einmal als Startpunkt gesetzt, von ihm aus wird dann in ähnlicher manier wie schon zuvor der nächste Punkt in J-Richtung gesucht und wenn nötig tritt auch hier nochmal die SaftyList Funktion in kraft um auch wirklich alle Punkte jeder Reihe ausfindig zu machen   (noch besser ausschmücken) |