A Relative Plane Algorithm

Jay Kraut

Abstract—**This paper describes a Relative Plane algorithm. It uses as input 2½D planes generated by an EM algorithm on a simulated 3D point field. The algorithm subdivides the map into groups of planes that are visible at the same time. Each group has a relative map calculated and these are combined into a global map. Rather than use current position, the last iterations untransformed observations are used for registration. The relative maps are calculated averaging the untransformed locations of pairs of planes. The pairwise comparison also leads to a method to identify the static edge of a plane when that plane is entering or leaving the viewpoint. The accuracy of this algorithm is dependent on the quality of the group selection. It is possible to adjust the groupings since the past untransformed observations are stored and available to recompute the relative maps. The simulation including the EM algorithm on points and the Relative Plane algorithm is able to run in real time.**

I. Introduction

One of the goals of robotics research is for a robot to be able to map an area without being given accurate position data. This is generally referred to as the simultaneous localization and mapping (SLAM) problem. The problem can be stated as: how does one place objects on a map if the current position is unknown or inaccurate and how does one know the exact current position if there is no map to reference?

This paper focuses on the SLAM problem using simulated vision input and no odometery data. There are many different approaches to this problem. Some of them are: Thrun et al. [1] who use the EM algorithm to convert a 3D point cloud into planes for texture mapping. Civra et al. [2] who use a one camera scale invariant feature transform (SIFT) from Lowe [3] and EKF for accurate outdoor navigation. Other use vision and occupancy grids such as the 2D grids [4][5] while some use 3D grids [6].

This work is inspired by observing indoor environments such as a hallway or an office environment. These environments can be modeled using planes. A plane can be defined by its normal and either its midpoint and size, or corner points. Planes though are not always viewed at their full size. As a plane enters the view it is growing and when it leaves it is shrinking. This causes the midpoint or corner points to have dynamic movement. An algorithm that can both filter noise and identify which corner points are valid is ideal. Preferably the algorithm would do both in an integrated fashion since this would reduce the computational complexity of having one algorithm for the filtering and a second to identify the valid corner points.

Regardless of the SLAM algorithm, planes or generally landmarks are first observed untransformed in the robot's viewpoint. That is, they are viewed from the robot's position being at coordinates (0,0,0) before being translated to a global position on the map. Most algorithms use a filtering algorithm to determine the current position as accurately as possible. Current position is then used to register the objects on the map.

There is another way to generate a map. Relative maps can be made of landmarks that are seen together. In a relative map, landmarks are mapped relative to a basis landmark. Using the untransformed observations of many iterations, a landmark's average relative distance to the basis object can be calculated. These relative maps can be merged together to form a global map. If the previous iterations untransformed landmarks are used to match the next iterations untransformed landmarks, current position generally does not have to be used.

Several papers use similar ideas in relative algorithms such as: Mei et al. [7] who use the concept of relative space to register the output of stereo cameras. Newman [8] stores landmarks in terms of relative position and has an algorithm to constrain the landmarks since there might be multiple links to each landmarks causing different global locations. Csorba [9] is perhaps the closest to this work as it links point based landmarks using a third point for relative angles. Lu et al [10] does scan matching between frames and stores all of the scans so that they could be combined to minimize error. Several papers use the concept of subdividing space such as: Lisien et al. [11] who use a layered approach combining EKF with topological maps. Frese [12] provides a framework called Treemap to subdivide an area into smaller ones based on which features are close together; and Pinies et al [13] who use a single camera and divides the features obtained into sub maps.

The Relative Planar SLAM algorithm presented here is tested in a simulation using as input 2½D planes generated by the EM algorithm [1] on a simulated 3D point field. A data structure, the *rltplane* is used to store a single planes untransformed observations throughout time. It is possible to derive the relative location and orientation of *rltplanes* that are visible in the same time interval calculating the average of the stored relative location and orientation. By grouping *rltplanes* based on visibility and chaining together groups based on common *rltplanes* it is possible to generate a map that is not dependent on maintaining current location as a

state. The accuracy of this algorithm is dependant on the quality of the group selection that maximizes the quantity of untransformed planes.

II. The Relative Plane Algorithm

The section describes the algorithm in steps: starting from registering a plane, to forming groupings, to pair wise plane comparisons and to calculating a map. Then the motion model is discussed which allows the algorithm to operate when only one partial plane is seen. The last section describes efficiency considerations in terms of groupings and comparison orders.

A. Registering planes

The algorithm receives untransformed planes as input from the viewpoint's perspective. Each planes location and normal is relative to the viewpoint being at location (0,0,0) and 0^0.

Each untransformed plane from the current iteration is matched against previously seen planes. When the algorithm starts there are no planes so each newly seen plane is placed in a data structure called *rltplane* shown in Figure 1. The data structure stores the untransformed observations by iteration. For the rest of this section *Rltplane* is used to refer to an instance of plane that has been seen many times from different viewpoints. Plane refers to a single observation of a plane.

In the next iteration, *rltplane*s are available to be compared to. The latest untransformed plane of each *rltplane* is used for matching. If an untransformed plane matches to a *rltplane* using the previous iterations untransformed plane, it is placed into the *rltplane* structure.

The Plane matching first compares the last seen iteration. Two different planes may look the same from the viewpoints perspective at different times so it is important to only use planes that have been seen in the last few iterations. The next test compares the planes equations and continues if they are within a constant value. Afterwards, the bounding boxes of each plane are compared to each other. Even if the bounding boxes overlap, the overlap must be sufficient otherwise two adjacent planes might incorrectly match to each other. If the plane matching passes all of its tests, the test gets a positive score based on how well it passed each test. One plane may match to more than one *rltplane* if the planes are close together. The *rltplane* with the highest score is the one the plane is registered to.

It is desirable to not have to compare a plane to every *rltplane* since that plane matching will be proportional to the total amount of planes on the map. Some mechanism should be used so only recently seen *rltplanes* are compared to. The current implementation compares a plane to only *rltplanes* from the last few seen groupings, and the ungrouped *rltplane* list.

When backtracking or closing the loop previously seen and mapped planes are observed again. In this case even though a plane has already been mapped, the registration using the previous iterations will fail to find a match. Before creating a new *rltplane* the untransformed plane can be transformed using current position and compared to the global map. As described later, current position is computed every iterations by comparing the current plane observations against the map. The map should always be locally accurate but if there are errors it may not be globally accurate. Backtracking should always work but closing the loop will only work if the global map is accurate to the actual map within the plane matching bounds. If a plane is successfully mapped to a previously seen *rltplane* the untransformed plane is placed into the structure. The untransformed plane matching will automatically work in the next iteration.

Iteration	Untransformed location	Untransformed plane equation
1	(x_1,y_1,z_1)	(nx_1,ny_1,nz_1,d_1)
2	(x_2,y_2,z_2)	(nx_2,ny_2,nz_2,d_2)
3	(x_3,y_3,z_3)	(nx_3,ny_3,nz_3,d_3)
…		
N	(x_n,y_n,z_n)	(nx_n,ny_n,nz_n,d_n)

Fig. 1. The rltplane data structure. This is a partial description as the full data structure contains other variables such the bounding box and the vertices.

B. Grouping planes

RltPlanes are grouped according to visibility. An example of the groups can be seen in Figure 2. In this completed map

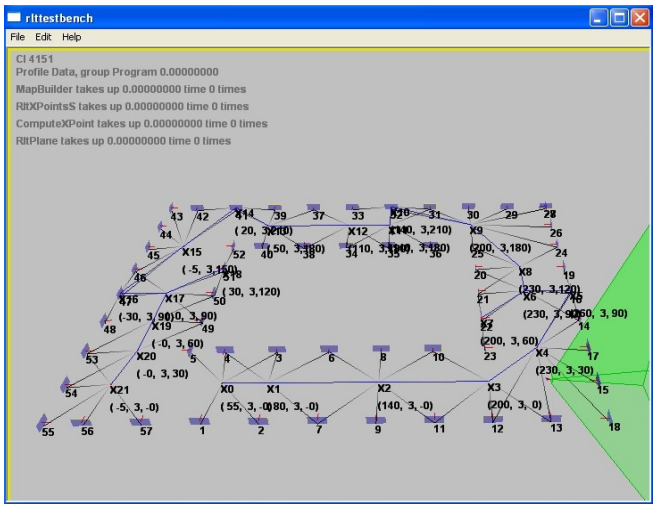

Fig. 2. A completed map. The lines from the planes signify which grouping the plane belongs to.

the *RltPlanes* are seen in blue with lines connecting to any grouping they are part of. The groupings are denoted with the letter "X" followed by a number.

When a new *rltplane* structure is created it is placed into the ungrouped list. This list is queried every iteration and *rltplane*s with sufficient observations are processed. If a *rltplane* contains below the minimum amount of observations, after a constant amount of iterations it is removed from the list.

A grouping contains only *rltplanes* that are seen in the same time interval. When a *rltplane* is first seen it may be difficult to judge which grouping it should belong to. The initial algorithm simply checks if an ungrouped *rltplane* is present in the same time interval as every other *rltplane* in a given grouping. If it is, the ungrouped *rltplane* is placed into that group and removed from the ungrouped list.

This initial grouping may not be the most efficient grouping in terms of the time interval available to calculate the relative map. Since *rltplanes* are placed onto the map after only a few iterations, it is not known how many total iterations that *rltplane* will have in common with the rest of the group. It can be possible to wait for more viewed iterations to add an *rltplane* to a group but then the *rltplane* will not be present in the map for those iterations. A better way it to check in the future how well the grouping is, and if required change it. This is described in a later section.

When a *rltplane* cannot be added to a previous grouping, a new one is created. The new grouping contains *rltplanes* from the previous grouping that are in the same time interval as the ungrouped *rltplane*. This is required since the grouping's relative map is to be linked to a previous grouping to form a global map.

C. Plane relative comparisons

Given two *rltplane*s that are present in the same time interval it is simple to calculate their relative locations. Some care most be taken though since the viewpoint may be moving and rotating. Figure 3 shows that using just the midpoint of two planes is translation invariant to movement of the viewpoint but not rotational invariant.

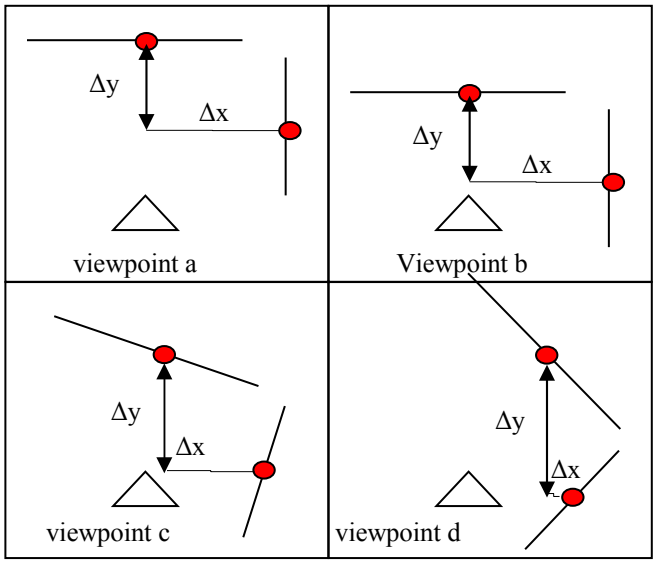

Fig. 3. The relative distance is translation invariant as seen in viewpoint a and b, but not rotational invariant as seen in viewpoint c and d.

To compare two planes, the normal of one plane is used to rotate both planes so that the first plane is on always on the same basis. For example, the first plane can be rotated so that it is always on the x-axis. Then every midpoint comparison will be rotation and translation invariant.

Since the comparison is rotation and translation invariant the average of the comparisons over the time interval where the two *rltplanes* are observed can be used to filter out noise.

Given two planes rltplane 1 and rltplane 2 the algorithm to solve the location average difference L_{av} (Δx, Δy, Δz) and average orientation average difference O_{av} over the interval where they are both visible is:

Initial $L_{av\ and}$ O_{av} to zero
For each iteration in the given interval
  Obtain the untransformed planes from each rltplane
  Rotate both planes so rltplane 1 is at 0 degrees
  Add the Δx Δy Δz to L_{av}
  Add the orientation difference to O_{av}
Divide $L_{av\ and}$ O_{av} by the size of the interval

The above algorithm works if the mid point is static, however it is not. When a plane is first seen in a viewpoint it is growing. When it leaves the viewpoint, as the viewpoint passes it by it is shrinking. If the midpoint is used for the comparisons it will cause a map error to occur. The growing and shrinking error, and the difficulty filtering it led to this algorithm. It was realized that the problem was not a filtering problem, rather it is an identification problem.

Each plane, as long as it is smaller than the viewpoint's viewing distance should always have at least one static edge seen. This static edge should always be identified and used for the comparisons instead of the midpoint. The static edge

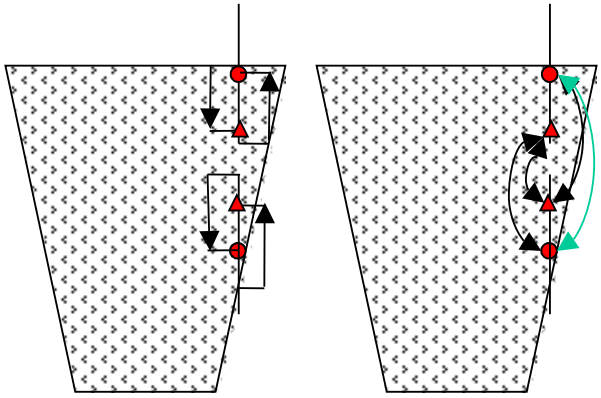

Fig. 4. The viewing frustums are the trapezoid shapes. There are two planes inside both only partially visible. Extending the plane size from the corner points (left figure) yields two correct midpoints (circle) and two incorrect ones (triangles). This forms 4 comparison pairs (right figure) with one is green being the correct pair.

should be identified at a minimum computation cost.

It is not desired to identify the edges using an algorithm that correlates one plane against many. Also using the average movement compared to an edge can be dangerous if there are only a few planes as the average may be wrong. The method used to solve this problem takes advantage of the fact that planes are compared to other planes in pairs.

Figure 4 shows the method used to identify static edges in pair wise comparisons. Figure 4 shows two planes and a viewing frustum where both planes are only partially seen, and the mid point is incorrect.

Referring to the left side of Figure 4, two assumed midpoints for each plane can be calculated by extending the maximum size of a plane along each of the edges. The circle denotes the correct midpoint and the triangle denotes the incorrect one.

Referring to the right side of Figure 4, each assumed midpoint can be compared against both midpoints from the other plane. This forms four comparisons pairs, which will all have an average distance over a time interval. It is possible to calculate the standard deviation for each of the four pairs. The pairing with the assumed midpoints from the static edges will have the lowest standard deviation. The edge that is static is saved since it is later used when linking the map and calculating current position.

It may be possible that there are two planes that are parallel but across from each other that will have two comparison pairs with a low standard deviation. In this case one of the pairs contains two static edges and the other two dynamic edges. Without other information it is not possible to know which one is static and which one is dynamic. To avoid this, a plane reordering process described in a later section rearranges the pairings used for the comparisons.

One issue that must be taken care of is that the static pairings changes depending if a plane is growing as it is first seen or shrinking as it leaves the viewpoint. Which edges are static changes as the viewpoint moves. The interval used to calculate the average and standard deviation must be split up if it is detected if a plane changes state from growing to shrinking.

The current implementation monitors the plane sizes to adjust the intervals. It may be also possible to monitor the output of the averages and identify the changing static edges that way. Also the current implementation uses assumed midpoints rather than using edge comparisons. It may be possible to instead use edge comparisons instead of assumed midpoint comparisons.

D. Relative and global maps.

Given a grouping, a comparison chain can be formed. The comparison chain links each *rltplane* in the group to every other *rltplane* in pair wise comparisons. Starting from one *rltplane* at position (0,0,0) every other *rltplane* has its relative position and orientation calculated to the first *rltplane*. This forms a relative map for each grouping from a basis *rltplane*. For display purposes the centriod of the group can be calculated and used for rendering purposes as shown in Figure 2.

The first observed iteration of the planes can be assumed to place the robot's viewpoint at (0,0,0) or some other inputted coordinate. The first grouping can be placed on the global map by comparing the relative maps coordinates of each *rltplane* to its first observation.

Every other grouping contains *rltplanes* that are present in previous groupings. These *rltplanes* can be used to translate the basis *rltplane* in a grouping from (0,0,0) relative coordinates to global coordinates. By processing each relative group using previously globally mapped *rltplanes*, a global map is created.

With a global map, current position is easy to calculate. The present observation's untransformed planes can be compared to the global map to calculate the current position. Current position is only used in a functional sense for backtracking and closing the loop.

It is possible to avoid any use of current position by instead always creating new rltplanes rather than use global matching. Once the *rltplanes* are placed onto the global map it can be checked to see if it should be merged with a previous *rltplane*. This should work with backtracking but it will only work closing the loop if the map is accurate within the plane matching bounds. The current implementation does not do this since it would likely use more overhead. However if current position is noisy or affected by dynamic data, it may be desirable to use this method.

E. Motion Model

Arriving at the end of the hallway and having only one plane visible can be a common occurrence encountered when traveling in a planar environment. This is shown in Figure 5, where this case occurs every time the robot turns. One

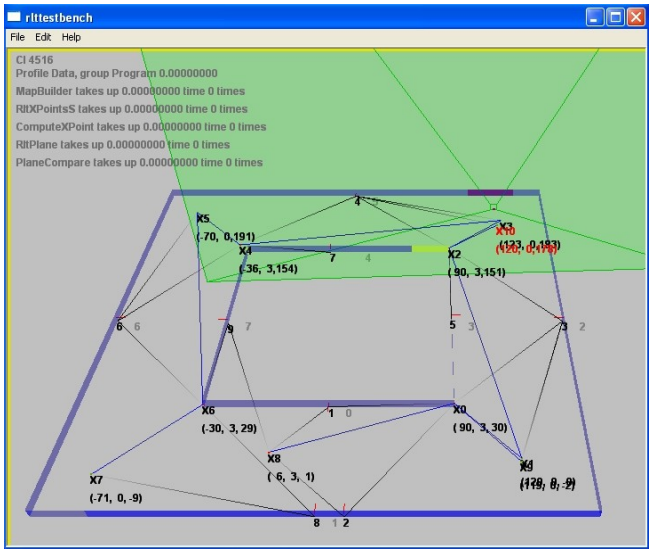

Fig. 5. Note in the top right only one partial plane is seen. The part of the plane that is shaded red is the only part visible in the frustum. At this point the robot is on its second time around the map.

solution might be to use the distance of point to plane equation to solve the current position. This unfortunately would lead to relying on using the previous location to calculate the current location, which is not ideal. There is a better solution than this that is similar.

The *rltplane* at the end of the hallway would be correctly placed on the map earlier when compared to *rltplane* passed by that are no longer visible. The problem therefore is not the placement of the *rltplane*. The only unknown information is the *rltplane's* maximum size as it expands as the viewpoint rotates. So the problem is not to calculate the current position, rather how to track how the *rltplane* expands as the viewpoint moves. Notice that this means that the *rltplane* orientation is not affected by any data given when the viewpoint is rotating since it is fixed to its position given by the rltplanes that have been passed by.

When the algorithm detects that only one partial *rltplane* is present it switches to the motion model. The maximum *rltplane* size is calculated as the robot's viewpoint moves, but the global position and orientation are not. As soon a second plane is observed the motion model ends.

F. Groupings and plane comparisons order

It is ideal to select the groupings and plane comparison order such that the largest possible time interval is available for the *rltplane* comparisons. Unfortunately when a *rltplane* is first observed, it is not evident what the best grouping might be. It is also not ideal to wait until the best grouping is known to place the *rltplane* into a group and onto a map.

The algorithm uses a regrouping and reordering process to optimize the intervals available for the comparisons. After a constant amount of iterations, a group is evaluated for reordering. Reordering looks at the *rltplane* comparison order and chooses the comparison order to maximize the total quantity of iterations used for the comparisons.

After another constant amount of iterations, a second evaluation occurs. The group is evaluated to find any *rltplane* that have a small comparison interval. *Rltplane* with an interval too small are removed from the group, and attempted to be placed into another group. If a *rltplane* cannot be placed into any group that would have sufficient iterations for that plane to be placed on the map, it is removed from the map altogether.

An algorithm that places objects into groups and may revaluate the groups in the future requires sufficient storage of past untransformed observations. These untransformed observations are used to recalculate a new grouping. The storage is required to be at least as big as it is possible to go back in iterations in the regrouping process.

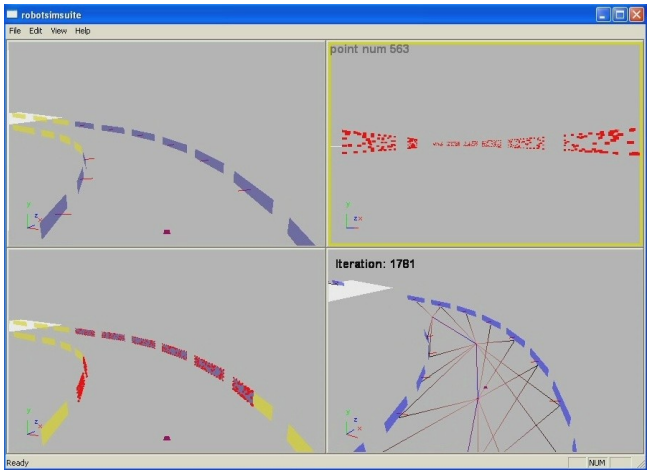

Fig. 6. An example of the simulation used to test the algorithm. The top left window shows the relative map in blue compared to the global map in yellow. The bottom left window shows the output of the EM algorithm compared to the input points. The top right shows the 3D points used as the input. The bottom right shows the connectivity of the relative map

III. Results

The results are shown in Figures 6,7 and 8. Figure 6 shows an example of the simulation used. The simulation uses 3D points as the input and those points are used to generate planes using the EM algorithm. Figure 7 shows the results of the simulation with no noise. The algorithm is able to reconstruct the map without any errors. Figure 8 shows the results of the simulation with noise being applied to each point. The noise used is sufficient so it is not possible to identify individual points. When the noise is further

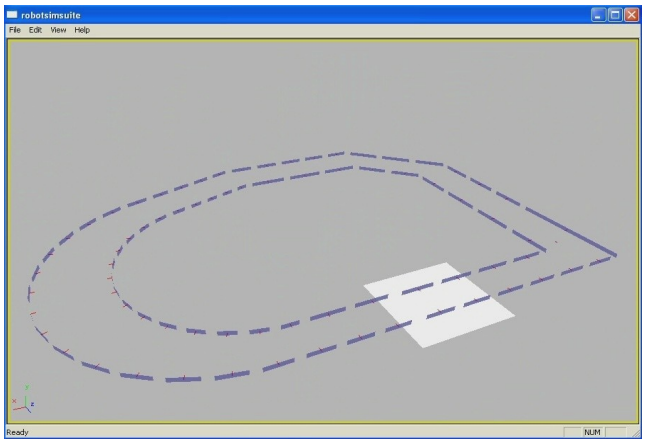

Fig. 7. Results of a simulation without noise.

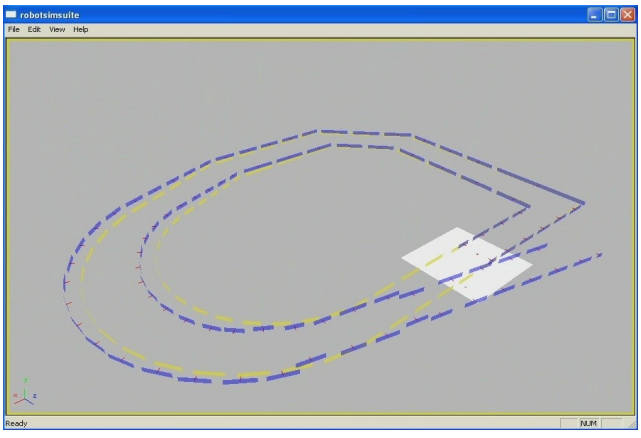

Fig. 8. Results of a simulation with noise.

increased in testing, it starts to become difficult for the EM algorithm to indentify planes.

The simulation including the EM algorithm to convert the points into planes runs in real time at 30 iterations a second on an AMD 64 3400+.

IV. Conclusion

The Relative Plane algorithm shows a lot of potential. Its main feature is that it is able to identify the static edges efficiently. It also has a motion model that works when only one partial plane is visible.

Any future testing on actual data requires segmented planes or a point cloud that can be segmented into planes with the EM algorithm. For the algorithm to work, the input data would have to be continuous, so there are enough observations to be able to identify static edges.

The Relative Plane algorithm does several things well: it can work with noisy input data, identify static edges, and it works in real time. It was realized that this algorithm using points rather than planes would have the same properties. This is discussed in future work.

V. Future Work

There are several implementation issues that can be improved upon. The current implementation is only 2½D. It would be desirable for the algorithm to be in full 3D. There are several places where some parts of the algorithm are not as computationally efficient as they can be.

There are also some architecture issues. One of the largest is that the groups are hard coupled to each other with linking planes. When a plane is removed from a group this link may break. The safest way to solve this is to have a roll back mechanism to roll back time, change the grouping and roll forward time. If the groups are soft linked this does not need to occur.

The current implementation has a limited mechanism for dynamic noise. It can remove planes that have limited calculation intervals but cannot deal with a plane that is moving around.

The Relative Point algorithm [14] shown in Figure 9 is adapted from this algorithm. All of the issues mentioned with the Relative Plane algorithm are fixed in its implementation. The Relative Point algorithm is proven to have a computation complexity of $O(n_a \log n_a)$ where n_a is the average quantity of observable points. For an average of 98 visible points per iteration, and a total of 1557 points, its average execution speed is 2 ms. The Relative Point algorithm uses the untransformed observations and a binning algorithm to identify dynamic points in O(nlogn). It is also shown to have comparable accuracy to an EKF in a 6D simulation.

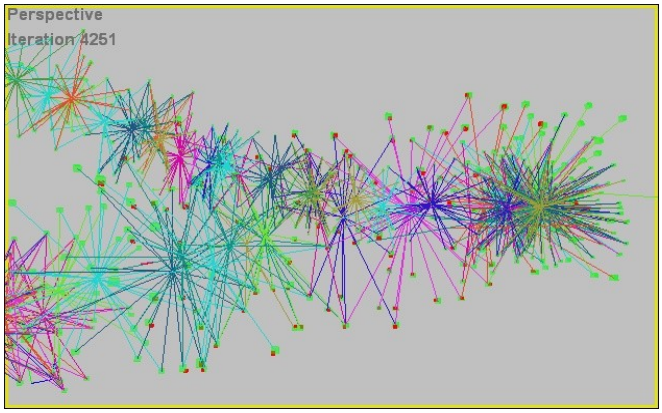

Fig. 9. An example of the Relative Point algorithm in a 6D simulation. Points are grouped into relative maps using their untransformed observations.

References

[1] S. Thrun, C.Martin, Y.Liu, D. Hahnel, R. Emery-Montemerlo, D. Chakrabarti, and W. Burgand. A real-time expectation maximization algorithm for acquiring multi-planar maps of indoor environments with mobile robots. *IEEE Transactions on Robotics*, 20(3):433-443, 2004.

[2] J. Civra, O. Grase, A. Davison, J. Montiel, 1-Point RANSAC for EKF-Based Structure from Motion. *IROS.* 2009

[3] D. Lowe, Distinctive Image Features from Scale-Invariant Key points, *International Journal of Computer Vision,* 60,2, pp. 91-110, 2004.

[4] P. Elinas, R. Sim, J. Little, sigmaSLAM: Stereo Vision SLAM Using the Rao-Blackwellised Particle Filter and a Novel Mixture Proposal Distribution. *International Conference on Robotics and Automation,* May 2006.

[5] D. Murray, J. Little, Using Real-Time Stereo Vision for Mobile Robot Navigation. *Autonomous Robots 8. p 161-171* 2000

[6] H. Moravec, Robot Spatial Perception by Stereo Vision and 3D Evidence Grids. *CMU Robotics Institute Technical* Report CMU-RI-TR-96-34 1996

[7] C. Mei, G. Sibley, M. Cummins, P. Newman I. Reid, A Constant Time Efficient Stereo SLAM System. *In Proceedings of the British Machine Vision Conference* September 2009

[8] P. Newman On the Structure and Solution of the Simultaneous Localization and Map Building Problem. *PhD Thesis University of Sydney* 1999.

[9] M. Csorba, Simultaneous Localisation and Map Building, *Phd Thesis, University of Oxford* 1997

[10] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots,* 4:333-349, 1997.

[11] B. Lisien, D. Morales, D. Silver, G. Kantor, I. Rekleitis, H. Choset, Hierarchical Simultaneous Localization and Mapping. *Proceedings of the 2003 IEEERSJ Intl. Conference on Intelligent Robots and Systems* Las Vegas, Nevada. October 2003.

[12] U. Frese, Efficient 6-DOF SLAM with Treemap as a Generic Backend. *2007 IEEE International Conference on Robotics and Automation,* Roma, Italy, April 2007

[13] P. Pinies, J. Tardos, Large-Scale SLAM Building Conditionally Independent Local Maps: Application to Monocular Vision, *IEEE Transactions on Robotics* V 24 I 5 Oct 2008.

[14] J. Kraut, A Relative Point algorithm, submitted to IROS 2011.