A Relative Mapping Algorithm

Jay Kraut

Abstract— **This paper introduces a Relative Mapping Algorithm. This algorithm presents a new way of looking at the SLAM problem that does not use Probability, Iterative Closest Point, or Scan Matching techniques. A map of landmarks is generated by using the average relative location difference between landmarks. This means the algorithm does not use any known, estimated or predicted movement or position data. In addition, the Relative Mapping Algorithm has the capability to identify dynamic landmarks using a binning algorithm. The algorithm is shown to have a fast constant time $O(n_a \log n_a)$ computation complexity where n_a is the average quantity of points that are visible. In limiting testing the accuracy of the Relative Mapping Algorithm is shown to be comparable to the Extended Kalman Filter.**

I. Introduction

The topic of this paper is the Simultaneous Localization and Mapping (SLAM) problem. Algorithms such as the Extended Kalman Filter (EKF) [1] with submapping [2] and FastSLAM [3][4] have been used to solve this problem. There has been similar work in the past with submapping [2][3][5] (and many others) and the use of relative location [6][7].

Most SLAM algorithms use movement or position data. This data can be generated by sensors (odometry) or estimated comparing a landmark's sensor data to the known landmark's position. This position data is then used to register future landmarks. That is, given a landmark obtained from a sensor reading, match that landmark with one already mapped. One potential drawback of using position data for landmark registration is that if there are large errors, landmarks will not be registered correctly. Large errors can occur if some landmarks have dynamic movement. A landmark is dynamic if its movement is not correlated to the movement of the robot.

Most SLAM algorithms first transform a landmark from the robot's relative space into global map coordinates for registration. The robot's sensor data is originally in the robot's relative space of the robot being stationary at position (0,0,0), and landmarks moving around the robot. After landmarks are segmented from the sensor data they are then transformed to global coordinates using current position. The registration continues with a nearest neighbor search to match the landmarks to the global map.

The Relative Mapping Algorithm uses a different approach to registration, and to SLAM in general. Landmark data is stored in a robot's relative space. Landmarks are only transformed to global coordinates in the last step. To register landmarks, new observations of landmarks in relative space are compared to the previous iteration of landmarks in relative space.

If a pair of landmarks are both static then their relative location difference to each other is consistent. The relative location difference of one landmark to another can be averaged over many iterations to filter noise. These relative location differences of pairs of landmarks can be combined into a global map. Current position can be calculated by comparing the current iteration's relative landmarks location to the global map. Note that current position is not used in any calculation and only used for display purposes.

This may appear to be similar to iterative closest point or scan matching but it is not. In those algorithms, an estimate of the movement or current position is first calculated. This estimate is used to translate the landmarks to global coordinates. Any error in this estimate is translated into the location of the landmarks and thus the filtering process. By the original relative locations, inaccuracies introduced from generating current position do not affect the algorithm. Similarly, this algorithm does not have a practical restriction on the noise such as being Gaussian that the EKF has, or the need to model the noise distribution like FastSLAM does. As long as it is possible to register landmarks in the current iteration, using the previous iteration's landmark relative locations this algorithm should work.

Another way of understanding how this algorithm works is by thinking about mapping a hallway using a pencil and paper. Assume a hallway consists of several doors that can be considered to be landmarks. As the person walks he/she sees the first two doors and gets a sense (average) of their relative location difference. Assuming the first door is at a starting position, say (0,0,0), the second door is plotted using the relative location difference from the first door. Then the person moves along and sees the second and a third door. Again the person gets a sense (average) of the relative difference. Since the second door is already mapped, the relative difference is used to map the third door from the second door's mapped global position.

The description of the Relative Mapping Algorithm in this paper consists of describing the key processes of this algorithm such as landmark registration, generating a relative map using a consistent basis, visibility interval group creation, and dynamic detection. The implementation is over 5000 lines of C++ code and cannot be fully describes using pseudo code or mathematically in the space available. However from a theoretical basis, this algorithm is not too much more than an efficient implementation of the above hallway example. From the hallway example, it should be evident that the Relative Mapping Algorithm works, as it is analogous to that process.

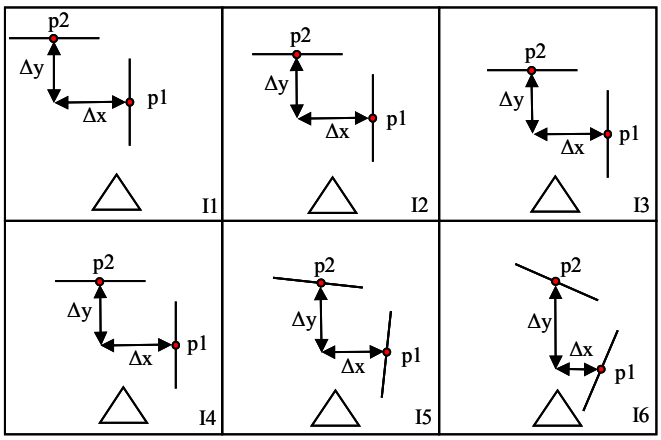

Fig. 1. Six iterations of robot movement. The triangle is the robot's viewpoint and p1 and p2 denote plane 1 and plane2. From I1 to I3 the robot has only translation movement so the relative Δx and Δy are the same. From I4 to I6 the robot rotates, notice that the relative Δx and Δy are no longer consistent.

II. The Relative Mapping Algorithm using Planes

The origins of the Relative Mapping Algorithm is in solving the SLAM problem for planes. Planes are different from points in that they have a midpoint, or corner points and a normal. The normal gives extra information that can be used, but the corner points (or midpoint) present a problem as they move as the plane expands or contracts. A plane expands as it is first seen and contracts as the viewpoint passes it by. This dynamic effect led to the need to use an algorithm that can identify if a corner point is static or dynamic and this is what led to the Relative Mapping Algorithm.

The first problem that was solved was how to map planes relatively. Figure 1 shows six iterations of a robot's movement. Notice that when the robot moves back and forth or side to side the Δx and Δy are consistent. However when the robot rotates the Δx and Δy are no longer consistent and are not usable.

Iteration	Untransformed location	Untransformed Orientation (normal)
1	(x_1,y_1)	(nx_1,ny_1,nz_1)
2	(x_2,y_2)	(nx_2,ny_2,nz_2)
3	(x_3,y_3)	(nx_3,ny_3,nz_3)
…		
N	(x_n,y_n)	(nx_n,ny_n,nz_n)

Fig. 3. A data structure that stores every iteration (observation) of an individual plane.

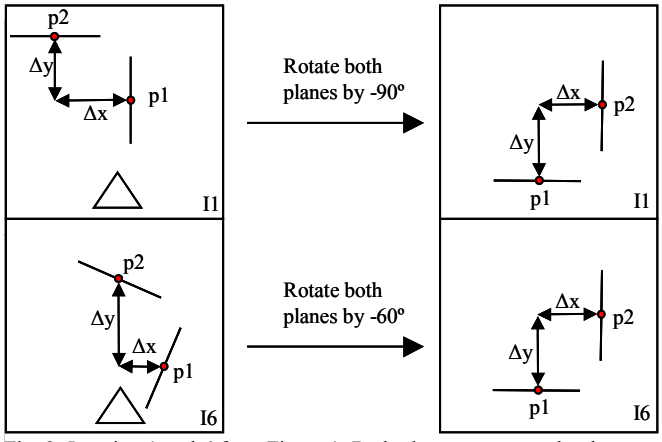

Fig. 2. Iteration 1 and 6 from Figure 1. Both planes are rotates by the same amount so that plane 1 is always at 0°. This creates a consistent basis so that Δx and Δy are now rotation and translation invariant.

To make the Δx and Δy consistent, a basis can be used. The basis is created by rotating plane 1 to be at 0°, and rotating plane 2 by the same amount. Figure 2 shows this process. By using a consistent basis, Δx and Δy are both rotation and translation invariant. They are consistent no matter where the robots viewpoint moves.

Before the average relative location of a pair of planes can be calculated, the planes have to be registered. The Relative Mapping algorithm registers a plane by comparing it to its previous untransformed location. The untransformed location is the location of a plane relative to the robot's viewpoint always being at location (0,0,0).

If the robot's viewpoint moves, the previous untransformed observation will be offset from the current untransformed observation by this movement. As long as this offset is small, a nearest neighbor can be used. All algorithms that do not have an independent odometry estimate are reliant on having a resolvable movement offset.

When the current iteration's observations are matched to the previous iteration's observations they are placed into a storage structure. This is shown in Figure 3. Many past observations are stored since the Relative Mapping Algorithm requires the use of past iterations of planes for its grouping process.

Given plane 1 and plane 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 plane
  Rotate both planes so plane 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 demonstrates calculating the average relative difference all at once. It is possible have a running

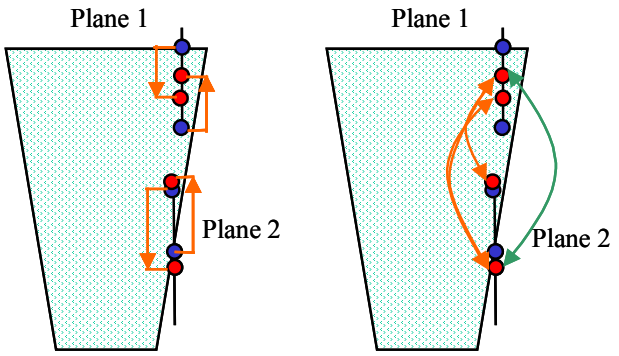

Fig. 4. Two planes (plane 1 and plane 2) are partially viewed as part of them falls outside the viewing frustum shown in the shaded trapezoid shape. The left figure shows the observed corner points in blue circles. For both corner points the size of the plane is offset to generate an estimated midpoint shown in red circles. In the right figure the four midpoint pairs are shown. One of the pairs will have a low standard deviation in time (shown as a green arrow) and this identifies the static corner points. The other pairs will have a high standard deviation.

average and only add the current iteration's relative difference to the running average and then divide.

The next problem the algorithm solves is how to identify dynamic corner points. Figure 4 shows a situation where two planes are observed but only partially as part of each plane falls outside the viewing frustum. For both of the two planes only one of the observed corner points is valid, as the other one is moving as the plane changes size. To identify the valid corner points estimated midpoints are created. These midpoints are generated by offsetting the plane's size from each plane's corner points.

Instead of only calculating one average distance, each of the four pairs has their average distance and standard deviation calculated over time. The pair with the lowest standard deviation is considered the valid pair of corner points.

The implementation of the plane version of the Relative Mapping Algorithm has considerably more functionality that

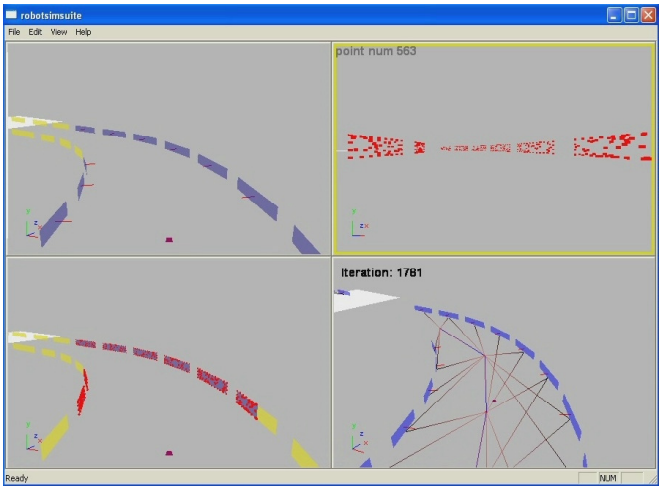

Fig. 5. Test bench used for the plane implementation. The top left shows the mapped planes compared to the actual planes. Top right show the robots perspective of seeing only points, Bottom left shows the results of the EM algorithm to form planes. Bottom right shows the full map with group connectivity

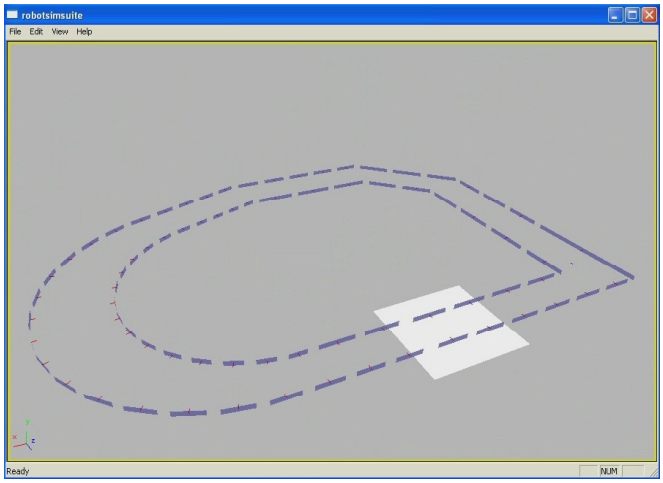

Fig. 6. Results without any noise, the map is generated perfectly

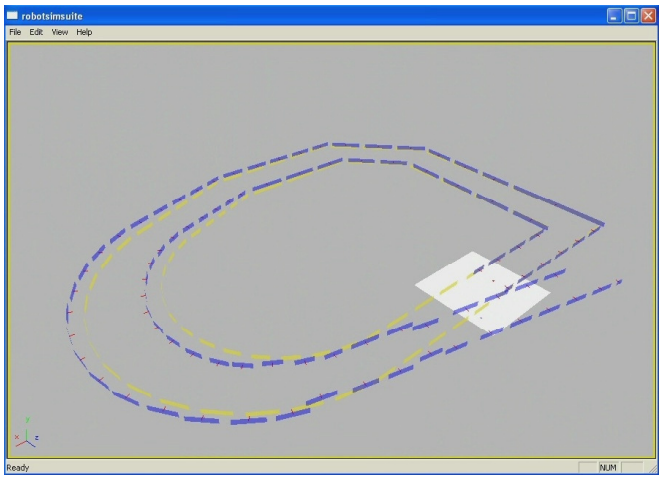

Fig. 7. Results with noise. The actual map is in yellow and the calculated map is in blue. The individual noise is set large enough so that points are no longer distinguishable from one iteration to the next.

described in this paper. Issues such as how to group the planes, interval splitting, and the motion model when only one partial plane is observed, are not discussed here. The implementation is described in more detail in [8] and pseudo code that requires multiple pages is available in [9].

The plane version of the Relative Mapping Algorithm is tested in a simulation. The input to the algorithm are points that are converted to planes using an EM algorithm [10]. The results are show in Figure 5,6 and 7. Figure 6 shows that without noise the algorithm is about to generate the map that contains partial planes and in one case only one partial plane without error. Figure 7 shows the results of a simulation at which the noise is set large enough so that individual points are not distinguishable from one iteration to the next. There is some error in the results but it is much less than if points are used as the input. A point based algorithm would not be able to work given the noise in Figure 7.

The plane implementation of the Relative Mapping Algorithm worked however there are several issues. One is that it only works in 2½ dimensions. There are also several

design issues. These can be fixed but there is a larger problem.

The larger problem is that segmenting planes is very difficult and this prevents the plane implementation from using real data. There is very little work on planar SLAM at this time. However, the algorithm itself is very interesting as it has good features such as it is fast, and not dependant on current position. The most interesting feature though is that it has some ability to identify dynamic features. Instead of working on the plane implementation further, the algorithm is adapted to using points.

III. The Relative Mapping Algorithm using Points

The plane version of the Relative Mapping Algorithm is adapted to use three dimensional points and to be used with six degrees of freedom robot movement. The registration is similar to the plane version as points are registered using the previous iterations untransformed or relative observations and stored in a data structure similar to Figure 3. Points are combined into groups where relative maps are created. These groups are combined to form a global map.

The relative map creation is similar to Figure 1 and 2 except that points do not contains orientation information so it is not possible to do pair wise relative differences. Instead at least three points are required. For efficiency, points are placed into groups where every point is visible in a minimum interval. Three points are chosen as a basis as shown in Figure 7. The basis points are chosen so that they are spaced apart in the group.

To create the consistent basis, the first point B0 is set to be at (0,0,0). The second point B1 is set to be on the negative x axis. The third point B2 is set to be on the xz plane. At the start of an iteration's relative map calculation, a transform matrix is created to place the three points on their designated positions. Then every point in the group is multiplied by this transformation matrix. This allows the

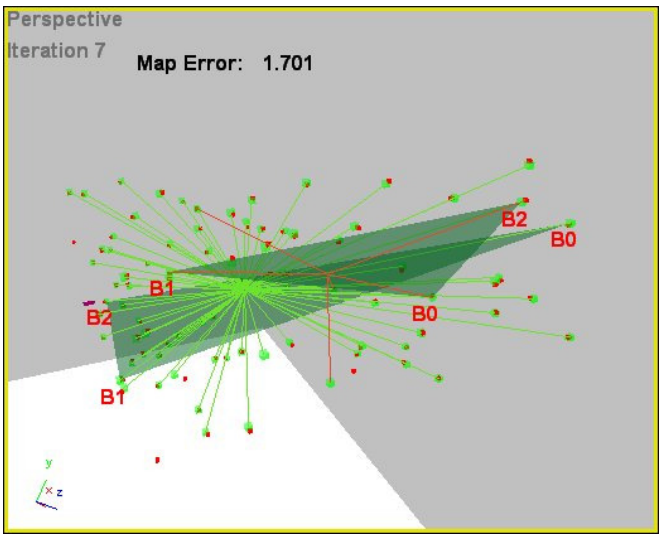

Fig. 8. There are two groups, one with green connecting lines and one with red connecting lines. Each group has three basis points denoted by B0, B1 and B2. These points form a plane that is consistent regardless of the robots viewpoint.

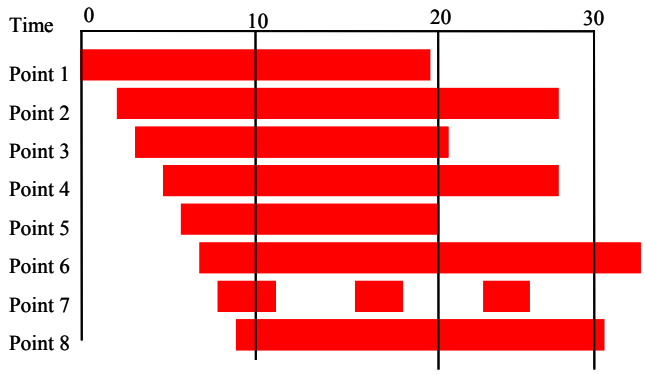

Fig. 9. The point charting structure. Each point, 1 through 8 is tracked if is it visible in a given iteration. This is used in the group creation process

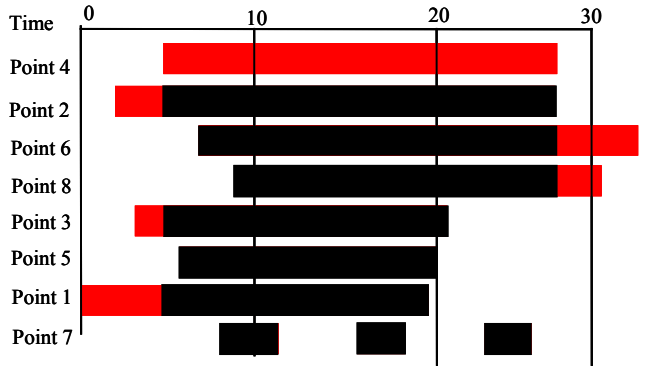

Fig. 10. The point chart is sorted so that points with more observations in point four's interval are in a higher position. Iterations that are in point four's observation interval are shaded.

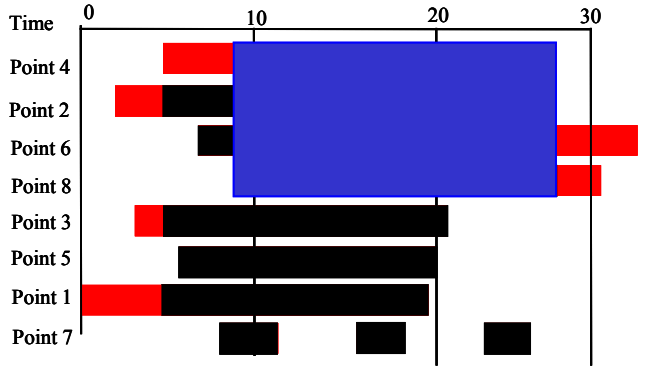

Fig. 11. A clustering algorithm is used to maximize the quantity of iterations available for the group to be calculated. The shaded blue section shows the interval where the first four points are all visible. A minimum of four points must be added per group, however additional points are added as long as long as the area of the shaded blue square (number of points multiplied by the common interval) goes up.

$(\Delta x, \Delta y, \Delta z)$ of every point to B0 to be consistent regardless of the robots viewpoint. The $(\Delta x, \Delta y, \Delta z)$ is then added to a running average. After dividing the running average by the number of observations, a relative map is created. In order to make the relative map consistent, every point in the group must be visible if an iteration can be used towards the average relative difference. The group creation algorithm is designed to ensure this.

The group creation uses a data structure that tracks every point that is visible in a given iteration. This data structure is shown in Figure 9, 10 and 11. To start the group creation

process, a point is chosen that has not been previously grouped. In the shown example this is point four. Then the point chart is sorted with points that have a larger interval in common with point four being higher in the charting structure.

All groups require a minimum of four points. Three points that have been previously been mapped and one new one. Additional points are added by traversing down the point chart structure. The next point in the structure is evaluated by first combining its observation interval with the current group observation interval. If the new interval multiplied by the total number of points in the group is higher than the previous total, the evaluated point is added to the group. If not the grouping algorithm is done.

A grouping process guarantees that each group has at least three points from a previous grouping that are already mapped. To align a group's relative map to the global map, the previously points in the group that have global locations are compared to their relative map location using Least Squared Fitting [11]. The first group is unique in that none of its points are mapped so they are Least Squared Fitted to their first observations assuming that the robot was at (0,0,0) at the first iteration.

There is robustness to the grouping algorithm. Notice in Figure 11 that point 7 which has large unobserved gaps has the lowest interval in common with point 4. The grouping algorithm can place every point into a group, but only points with large intervals are used as previous points in following groups. This means that only points with the largest intervals are used to propagate the map forward. Points with small intervals such as point 7 are not used to propagate the map forward and do not affect the overall accuracy of the map.

It is possible to run the grouping process several times. The first grouping can be performed after a few iterations to place points onto the map. The second grouping can be performed after more iterations where there are enough observations to run dynamic detection. The third grouping can be performed after points are no longer visible, at which time their total observation interval is known.

A point's global location is calculated using its highest level grouping. When all the points in a lower level group are mapped in higher level groups, that lower level group can be removed. This gives a balance between creating groups quickly and having the groups be as accurate as possible as more observations are seen.

When a point is grouped for a second time, that group's relative map is recalculated from the start of the group's observation interval. This iteration has long since past, but it is possible to be calculated since every point's untransformed reading are stored in a data structure (Figure 1). This data storage also makes it possible to identify dynamic points in O(nlogn) time.

Without storing past relative observations, the identification of dynamic points can require updating a correlation matrix. Every point can be compared to every other point, and points with a high correlation can be grouped together. Since it is not known in advance which

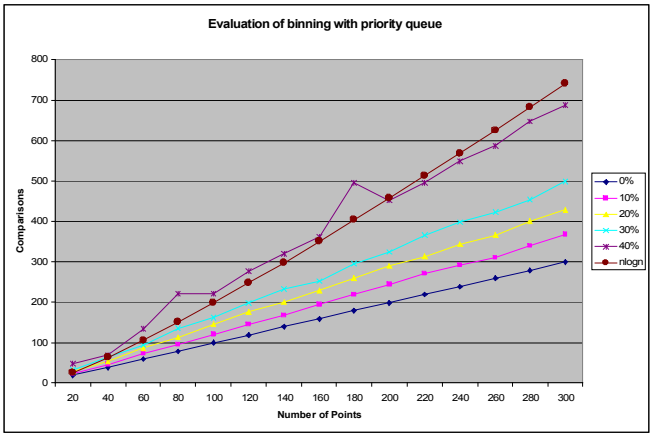

Fig. 12. The number of comparisons required versus the number of points. Even with 40% of the points being dynamic the number of comparisons is at or below the nlogn line. The binning algorithm become n^2 when approximately 47% of the points are dynamic.

points are correlated, every point needs to be compared against every other point. This is an $O(n^2)$ process.

The Relative Mapping Algorithm runs dynamic detection after many iterations in the second group creation. It is only possible to do this since the point's have their past observations stored. Rather than compare every point against every other point, points are placed into bins. Bins are then compared to each other using only one or a few of the points present in it. Two heuristics are used: first, when half the points are in one bin, that bin is the static point bin, second, a priority queue is used for the bin comparisons. With these heuristics the dynamic detection can be ran in O(nlogn) time as shown in Figure 12. Once a point is identified as dynamic it is no longer used in the map but it is still tracked for display purposes.

The dynamic point detection removes points with a high standard deviation. The group creation algorithm only uses points with large intervals as previous points. The combination of the dynamic detection and the group creation creates a robust point selection algorithm where poor points do not affect the accuracy of the map.

What happens when a point that is already mapped being observed after many iterations of not being seen? The registration that uses the previous iterations observations will not be able to register this point since it has not been seen for a while. Perhaps current position can be used to translate the point to global coordinates to find a global match. This is not ideal since current position may have noise. Current position is calculated the same way as the plane implementation, by using the current relative observations compared to their global mapped locations.

A better way is to add the point as a new point and place it into a new group. After several iterations, its global mapped location can be compared to the map to see if there is a match. If there is a match, the point can be merged with the matched point. The merging process is simple as the data storage in Figure 1 is copied over. This merging process allows the avoidance of the use of current position that may be noisy in any given iteration. The Relative Mapping

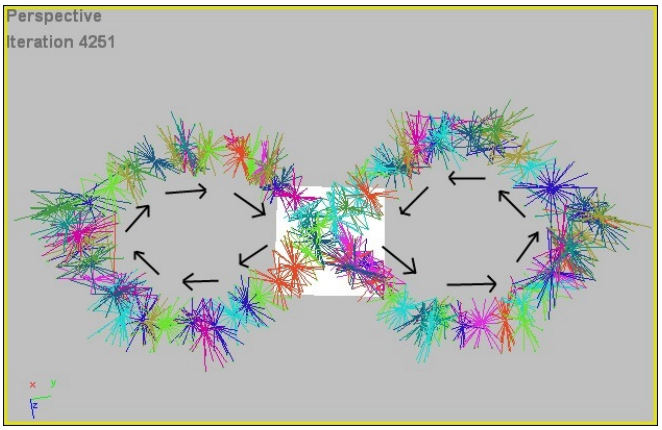

Fig. 13. Simulation used for testing, a figure eight. The path has a gradual incline and then a decline, so the robot returns to the middle only after a full loop

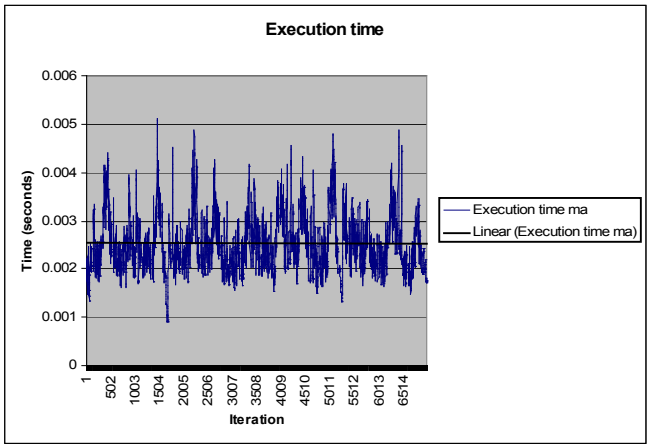

Fig. 14. Run time in seconds of the algorithm as it does one loop of the figure 8. It starts with 0 points and ends with 7400. Notice the run time is constant.

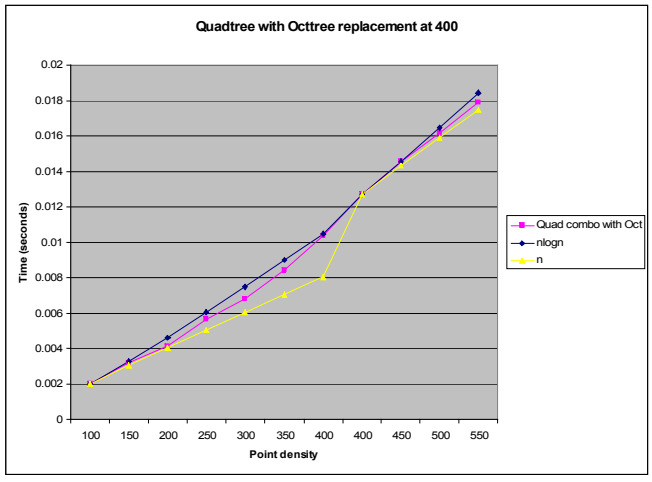

Fig. 15. Run time as the average quantity of visible points goes up. A quadtree is used for the point matching up to the 400 point density where it is replaced by an octtree. A kd-tree was also tested but was found to be slower.

Algorithm does not use current position or movement, estimated, predicted or known except for display purposes. The merging process can automatically closes the loop, if the map's accuracy is within the point matching bound.

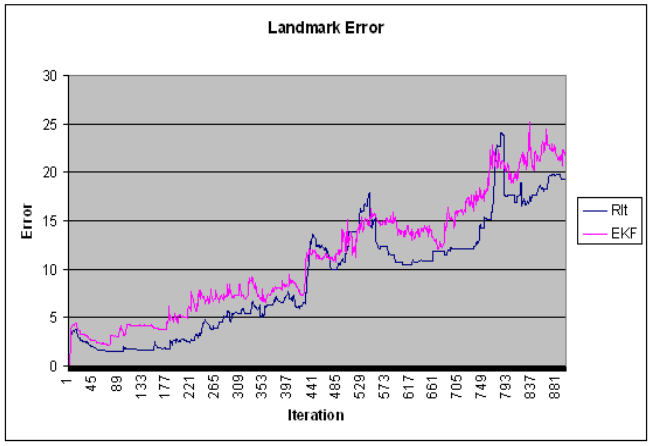

Fig. 16. Landmark error of the EKF versus the Relative Mapping Algorithm.

There are many more details to the implementation of the point version of the Relative Mapping Algorithm. Only the major topics are covered in this paper. Additional details are in [12] and a full description is in [9]. The implementation uses approximately 5000 lines of C++ code and cannot be easily described in a mathematical description or pseudo code. Part of the reason why it is so many lines of code is that the implementation is optimized for run time performance. The algorithm is best described using its roots in software engineering by describing the object oriented classes that are used in the algorithm [12]. The implementation resembles an optimization algorithm such as Binary Space Partition [13] closer than it resembles any current SLAM algorithm.

IV. Run Time Performance

The Relative Point algorithm is tested in a simulation of a figure eight path shown in Figure 13. The run time is shown in Figure 14. Figure 14 shows that the algorithm executes in constant time. Figure 15 shows the results of increasing the average quantity of visible points. The algorithm's computation increases within $O(n_a \log n_a)$ where n_a is the average quantity of visible points, as long as the data structure that does the point matching is chosen correctly.

V. Accuracy Comparison to EKF

The Relative Point algorithm is tested against a 6D no odometry EKF [14] obtained from the Mobile Robot Toolkit [15]. The point density is reduced to 25 points per 100 units length to allow the EKF to run. The observation noise is white Gaussian noise.

Figure 16 shows the landmark error for 900 iterations. The two algorithms have similar accuracy. If the noise model used is changed to flat white noise the Relative Mapping Algorithm outperforms the EKF by a wide margin.

Unfortunately at about 900 iterations, the EKF run time was approaching 1 second per iteration and slowing down further, so the testing stopped. The Relative Point algorithm run time was about 1 millisecond per iteration.

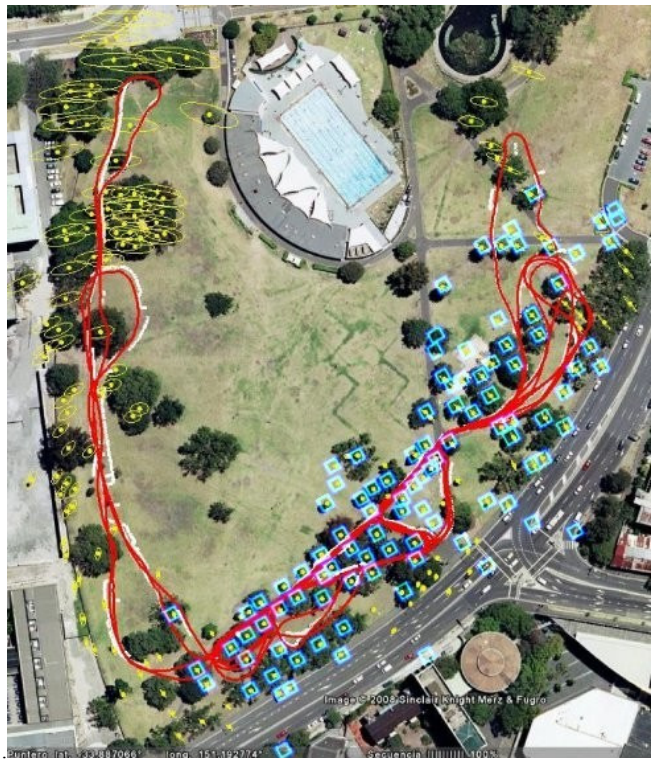

Fig 17. Landmark positions from the EKF in yellow dots and from the Relative Mapping Algorithm in blue squares.

Unfortunately due to the lack of any motion model the point implementation of the Relative Mapping Algorithm cannot be fully tested on the Victoria Park data set. This is due to the data set having many intervals where the quantity of visible points falls below four. The point implementation is designed to work with vision data that should always have a sufficient quantity of points visible. However there are two intervals, from 1181 to 1642 and 4220 to 4634 that have at least the minimum number of points visible.

Figure 17 shows the results of iteration 1181 to 1642. The Relative Mapping algorithm results are shown in the blue squares compared (manually stretched) to the results from [16]. It is not valid to do an exact comparison due to the short size of the interval used. However the results do show that the landmark locations of the Relative Mapping Algorithms are consistent with results from the EKF.

The average runtime of the Relative Mapping algorithm on the Victoria park data set is 0.00057 seconds. All testing performed using an AMD 64 3400+.

VI. Results

The Relative Point algorithm has a worst case computation complexity of $O(n_a \log n_a)$, where n_a is the average quantity of points visible.

The Relative Algorithm is able to identify dynamic points in O(nlogn) time. It is shown working in the figure eight simulation with 40% of the points being dynamic.

The accuracy of the Relative Point algorithm is compared to a 6D EKF implementation that does not use odometery. The accuracy of the Relative Point algorithm is shown to be comparable to the EKF. There is also a limited comparison using a subset of the Victoria Park data set, which the results are consistent with the results using an EKF.

VII. Discussion

The Relative Mapping Algorithm is perhaps more similar to optimization algorithms such as Binary Space Partition than it is to other SLAM algorithms. It has very beneficial properties in terms of computation complexity and speed, and the ability to identify dynamic points. Its accuracy has been shown to be consistent with the EKF in very limited testing. It can be said that given dynamic noise the Relative Mapping Algorithm will continue to work whereas algorithms without this capability will not. However it is too early and the testing is too limited to make a general statement about its accuracy.

Perhaps the most important result of this work is that it proves that SLAM can be done without making use of any movement or location data, and that SLAM can be solved using Software Engineering optimization techniques.

References

[1] M. Dissanayake, P. Newman, S. Clear, H. Durrant-Whyte, M. Csorba, "A Solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation* V 1713 Jun 2001

[2] L. M. Paz, J. D. Tardos and J. Neira, "Divide and Conquer: EKF SLAM in O(n)," *IEEE Trasnactions on Robotics.* Volume 24, No. 5, October 2008.

[3] S. Thrun, W. Burgard, D. Fox. "Probabilistic Robotics," *MIT PRESS*, Cambridge, MA, 2005.

[4] T.D. Barfoot, "Online visual motion estimation using FastSLAM with SIFT features," IROS 2005

[5] 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 Intellgent Robots and System* Las Vegas, Nevada, October 2003.

[6] M. Csorba, "Simultaneous Localisation and Map Building," *PhD Thesis,* University of Oxford, 1997.

[7] P. Newman, "On the Structure and Solution of the Simultaneous Localisation and Map Building Problem," *PhD thesis*, University of Sydney, 2000

[8] J. Kraut, "A Relative Plane Algorithm", Technical Paper, *www.jkrobots.com*, 2011

[9] J. Kraut, "The Development of a Relative Point and a Relative Plane SLAM Algorithms", *PhD thesis,* University of Manitoba, 2011

[10] 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.

[11] K. S. Arun, T.S Huang, and S. D. Blostein, "Least square Fitting of two 3D point sets," *IEEE Transaction on Pattern Analysis and Machine Intelligence,* 9(5) 1987 pp. 698-700

[12] J. Kraut, "A Relative Point Algorithm", Technical Paper, *www.jkrobots.com,* 2011

[13] Binary Space Partitions, *http://en.wikipedia.org/wiki/Binary_space_partitioning*, 2011

[14] J. L. Blanco, "Derivation and Implementation of a Full 6D EKF-based Solution to Bearing Range SLAM," *Technical Report, Perception and Mobile Robots Research Group,* University of Malage, Spain, 2008

[15] Mobile Robot Toolkit 6D-SLAM, *http://www.mrpt.org/6D-SLAM,* retrieved January, 2011

[16] P. Rodriguez, *http://webdiis.unizar.es/~ppinies/home.html*, 2011