AD

AD-E403 613

Technical Report ARWSE-TR-14009

ALGORITHMIC APPROACHES FOR PLACE RECOGNITION IN FEATURELESS, WALLED ENVIRONMENTS

Naomi Zirkind

January 2015

U.S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND ENGINEERING CENTER

Weapons and Software Engineering Center

Picatinny Arsenal, New Jersey

UNCLASSIFIED

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-01-0188*

1. REPORT DATE (DD-MM-YYYY) January 2015	2. REPORT TYPE Final	3. DATES COVERED (*From - To)*

4. TITLE AND SUBTITLE ALGORITHMIC APPROACHES FOR PLACE RECOGNITION IN FEATURELESS, WALLED ENVIRONMENTS	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHORS Naomi Zirkind	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army ARDEC, WSEC Tactical Effects, Protection & Interactive Technologies Directorate (RDAR-WSH-N) Picatinny Arsenal, NJ 07806	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army ARDEC, ESIC Knowledge & Process Management (RDAR-EIK) Picatinny Arsenal, NJ 07806-5000	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) Technical Report ARWSE-TR-14009

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report gives a detailed presentation of three algorithms for automated place recognition by a mobile robot in featureless, walled environments - where conventional feature descriptors are ineffective. The robot is assumed to have on it an inertial measurement unit (IMU) and a ranging device such as a light detection and ranging system. The environmental descriptors presented here are based on trajectory features and on junction features. The report shows how to derive the descriptor values at each point in the trajectory and how to compare descriptors recorded at different points in the trajectory to determine how closely they match. The output of the algorithms can be fed to an external mapping program and can be used to help correct errors in the map due to IMU drift. These algorithms can be combined with algorithms for fully featured environments to implement a nearly all-environment mapping system.

15. SUBJECT TERMS
Loop closure Place recognition Featureless environment Feature descriptor Mapping algorithm

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBE PERSON Naomi Zirkind
REPORT U	b. ABSTRACT U	c. THIS PAGE U	SAR	75	19b. TELEPHONE NUMBER (Include area code) (443) 861-1438

CONTENTS

FIGURES

FIGURES
(continued)

FIGURES
(continued)

TABLES

ACKNOWLEDGMENTS

The author acknowledges several technical discussions with Joshua Wainer of Robotic Research, LLC in connection with the algorithms presented here. These discussions helped refine the ideas that led to the composition of these algorithms. He supplied two images in this report.

BACKGROUND

Robotic mapping is an important capability for a variety of purposes. Robots can explore areas that are too dangerous or contaminated for humans to explore. People can use maps that the robots make of such areas to plan missions and activities to be performed in these areas. For example, in law enforcement and military applications, a robot could be used to explore and map a structure in preparation for entrance into that structure. Robots could also be used to explore structures contaminated with radioactivity or other contaminants.

As a robot travels through a new environment in order to map it, the robot must continually update the map as it gathers new information. In addition, it may encounter objects or locations of interest as it travels and must record their location for future reference. Thus, the robot must simultaneously map the environment and localize itself within that environment in order to record the location of places of interest. The discipline of simultaneous localization and mapping (SLAM) has been studied intensively over the past several years.

Many technical approaches have been proposed to the SLAM problem, each using a particular set of sensors on the robot. The simplest method for a robot to localize itself is to use a receiver for a global navigation satellite system (GNSS), such as global positioning system (GPS), Globalnaya Navigatsionnaya Sputnikovaya Sistema (GLONASS - a Russian satellite-based navigation system), Galileo, and/or Bei Dou. In addition to GNSS receivers, commonly used sensors are visual cameras (refs. 1 and 2), light detection and ranging (LIDARs) (refs. 3 and 4), inertial measurement units (IMU) like gyroscopes and accelerometers, odometry sensors, and even sometimes magnetometers (refs. 5 through 7) and barometers (ref. 1).

In some environments, such as indoor environments, GNSS signals are not available. These environments are sometimes referred to as GPS-denied. In such environments, the robot must rely on its onboard sensors in order to localize itself and make an accurate map. Generally, IMU signals are fused with camera and/or LIDAR data to derive an estimate of the robot’s location at any given time. However, noise in IMU signals causes the calculated robot position to drift over time. This cumulative drift error results in erroneous maps. One type of map error is incorrect loop closure, i.e., when the robot returns to a place it has previously visited, it “thinks” it is in a different location. For example, if the robot drives in a large circular path and returns to its starting point, the calculated trajectory will not be closed, though it should be. Figure 1 shows a conceptual illustration of how a trajectory calculation error can lead to an incorrect loop closure.

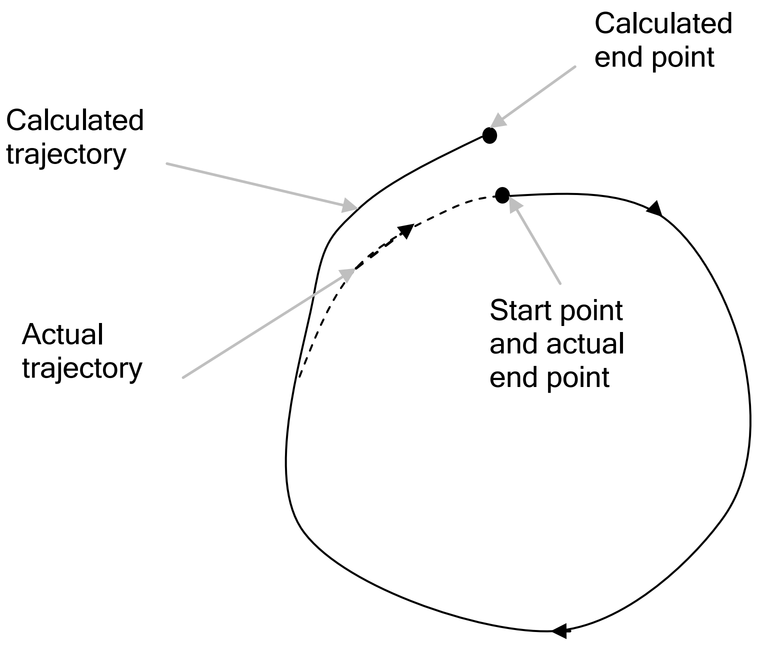


Figure 1
Conceptual diagram showing incorrect loop closure due to drift

One way to correct drift error is to recognize, using sensors besides the IMU, when the robot is revisiting a place that it had previously visited. If the robot could recognize a previously visited place, then it could inform the mapping system to close the loop and to close the trajectory so that both the previous and current visits are denoted by the same point on the map.

How can the robot recognize a revisit to a place? This question is referred to as the place recognition problem and is sometimes referred to as the loop closure detection problem. In general, this problem is addressed by defining some type of feature descriptor and cataloging the descriptor content of each place that is visited. Every time a new place is visited, the feature descriptor (or set of descriptors) is calculated for that place. The newly calculated set of descriptors is then compared with previously measured descriptor sets in search of a match. If a tentative match is found, the quality of the match is determined and compared to a threshold value in order to make a decision whether or not this is indeed a previously visited place.

If a place recognition is made, then the two points (the originally and currently calculated locations) are passed to a trajectory modification program. The trajectory modification program adjusts the trajectory so that it smoothly connects the originally and currently calculated locations. This trajectory correction completes the map correction that is performed as a result of the place recognition (ref. 8).

A variety of feature descriptor types for various types of sensor data have been proposed, e.g., normal aligned radial features (ref. 3), speeded-up robust features (ref. 9), surface entropy (ref. 10), normal distribution transform (ref. 11), and signatures of histograms (ref. 12). In order for a descriptor to be effective in characterizing a particular type of environment, that environment must have some uniquely identifiable features that can be represented in terms of that descriptor. For example, in order to use an edge detector, the environment must have edges in it, and different locations in the environment must have unique patterns of edges. That is the only way a particular location can be reliably recognized upon a revisit.

If the robot uses IMUs with cameras and/or LIDARs to map a featureless environment such as a network of hallways or tunnels, then no currently used feature descriptor can reliably characterize locations in the environment. This work presents new types of feature descriptors that can be used in such environments to recognize places. These descriptors can be used in featureless environments with hallways that have nominally parallel walls.

This work complements other work on indoor and outdoor mapping in GPS-denied environments. Some sensors and descriptors work better indoors, and some work better outdoors, but so far, to the author's knowledge, very little has been published on an effective and reliable descriptor for the featureless hallway or tunnel environment.

Reference 13 does present an approach for using trajectory features for place recognition, which is similar to the approach of the first of the three algorithms presented here. The approach of reference 13 characterizes "regular patterns" in the trajectory and tries to recognize previously traveled trajectory segments with the same pattern. That type of feature does not clearly distinguish between different but similar trajectory segments and can lead to many false positive judgments. In the experiment, six out of thirty loop candidates were incorrect. The approach presented here for trajectory based place recognition seems that it would more unambiguously recognize corners in the trajectory. However, the weakness of the approach in reference 13 is also its strength – it can recognize features in many different types of paths, not only paths with corners in them. Thus, that approach is in a sense complementary to the approach presented here, which clearly identifies corners, but recognizes only corners.

The work presented here could be combined with currently available place recognition approaches to create an (almost) all-environment mapping system. The reason for saying "almost" is that there is one type of environment in which place recognition does not seem to be feasible using just optical and/or LIDAR sensors, and that is a large, empty, featureless setting. In such a setting, perhaps a magnetometer or other type of sensor could be used together with inertial sensors for improved place recognition.

SUMMARY

The problem that the algorithms presented here address is how a robot equipped with an IMU, and possibly a LIDAR sensor, that is travelling in a walled, featureless environment can recognize a place that it has previously visited during the current excursion. This problem is important for robots that are mapping such environments because accurate place recognition helps the mapping algorithm to partially correct errors in the map that are due to drift in the IMU data.

This report presents three algorithms that use LIDAR and IMU data to extract relevant information about the environment in order to form descriptors of each visited place. Then, each time the robot visits a new place, it compares its newly measured descriptor with a selected set of previously measured descriptors to determine whether a place recognition has occurred. If the algorithm determines that place recognition has occurred, it passes the descriptor of the newly visited place, the descriptor of the previously visited place, and a matching score to an external mapping program.

The main novelty of this work is the formulation of descriptors for such apparently featureless environments. The descriptors are based on detected corners and junctions in the hallway structure in which the robot is travelling. In addition, this work presents techniques for comparing different instantiations of the descriptors to determine how well they match.

Although many algorithms have been published on place recognition in feature rich environments, to the author's current knowledge, place recognition in featureless, walled environments has not been addressed in the published literature. This work could significantly expand the set of environments in which automated place recognition can occur. However, due to programmatic constraints, these algorithms have not yet been implemented in computer code, so their true utility has yet to be discovered. The author recommends that these algorithms be implemented, and refined and revised if necessary, in order to enable robots to effectively explore new regions.

INTRODUCTION

The algorithms presented here are useful for place recognition and loop closure detection in situations in which a robot is driving in a network of possibly featureless hallways with nominally parallel walls. The purpose of this report is to present a detailed description of the algorithms so that other researchers could implement them or some variant of them and/or build on them to give them more capabilities.

This report presents the concepts of the three algorithms using verbal descriptions, diagrams, and flowcharts to illustrate the concepts. The algorithms will henceforth be referred to as (1) the trajectory based algorithm, (2) the LT-junction based algorithm, and (3) the general junction based algorithm.

The next three major sections describe the concepts, as well as the advantages and disadvantages, of each of the three algorithms. The first major section describes the trajectory based algorithm. This algorithm is useful for place recognition in situations in which a robot is driving in a network of featureless hallways that constrains its motion to either nominally straight line driving (with some meandering, possibly) or turning from one path segment onto another. The ground surface on which the robot drives is assumed to be nominally flat, and therefore, the environment for all three algorithms is represented as a two-dimensional (2D) world.

This algorithm is the most versatile of the three since it does not explicitly assume anything about the walls other than that they constrain the robot to move in fairly straight line path segments with well defined turns. However, consideration of the characteristics of junctions in the path as defined by wall contours can greatly assist in place recognition. Consideration of such junctions will add some complexity to the algorithm and is addressed in the two junction based algorithms.

The second major section describes the LT-junction based algorithm. The algorithm has this name because it detects junctions that have two legs – denoted as L-junctions and those that have three legs – denoted as T-junctions. This algorithm treats a junction shaped like a plus sign (+) as two back-to-back T-junctions. This algorithm uses more environmental information than the trajectory based algorithm. However it could be further improved by considering junctions with arbitrary numbers of legs.

The third major section describes the general junction based algorithm. This algorithm can detect junctions with arbitrary numbers of legs but requires an additional preprocessing step for the input data that the LT-junction based algorithm does not have. It requires a 2D visual image of the area that has been traversed so far to be produced.

Each of the three major sections first describes the overall concept of the presented algorithm. The section after that describes the type of data that is input to the algorithm and preprocessing steps that are required to enable derivation of the descriptor values from the input data. The next section presents the feature descriptor that is used to represent the key information about visited places and shows how it is derived from the preprocessed input data. The section after that describes the process for comparing descriptors to determine how well they match each other. The final section describes the derivation of the output data to pass to a trajectory adjustment program, which uses the place recognition information to recalculate the previously calculated trajectory.

METHODS, ASSUMPTIONS, AND PROCEDURES

This section describes in detail the trajectory based algorithm, the LT-junction based algorithm, and the general junction based algorithm for place recognition in walled, featureless environments.

Trajectory Based Algorithm

This section describes, for the trajectory based algorithm, the input data and its preprocessing, the feature descriptor and its derivation from the preprocessed input data, the descriptor comparison process, and the derivation of output data.

The trajectory based algorithm uses as simple and broadly applicable an approach as possible. This approach uses features of the trajectory itself to represent visited places. That is, at each increment of the robot’s travel, the algorithm uses a sliding buffer to determine whether the robot has made a turn. It does this by calculating a straightness score for all the poses in the buffer. The score is based on the mean squared deviation of all the poses from the best fit line. A large straightness score means a nonstraight line.

A corner is detected when the straightness score increases to a peak and then decreases again. If the peak is higher than a user-defined threshold, then it is accepted as a corner. The threshold prevents gentle curves in the trajectory from being considered as corners. Since we are looking for maxima in the straightness score, we cannot immediately determine whether we are at a corner. We must travel some distance *beyond* the corner in order to determine whether we have traversed a corner. The situation is analogous to the way economists cannot determine whether we have entered or recovered from a recession until several months *after* the event since only then can they identify patterns in prior data.

A database of detected corners is maintained, and new candidates are compared with detected corners according to some similarity metrics. The highest scoring candidate, if its score is above a user-defined threshold value, is considered a recognized place. If a place recognition is made, the trajectory is to be updated (by a separate program) to indicate that the actual corner location is the location that was estimated in the original visit to that corner.

Input Data and Preprocessing for Trajectory Based Algorithm

The input data for the trajectory based algorithm is a time stamped sequence of 2D robot poses. Each pose is of the form (x, y, θ, t), where x and y are the 2D coordinates of the center of the robot projected onto the ground in a global frame of reference, θ is orientation angle of the robot, and t is the time at which the pose data was measured. As the robot drives along, the pose is measured and recorded at regular intervals, resulting in a continuous stream of pose data.

A moving buffer is used to store the most recently recorded set of consecutive pose values, denoted by p_i, where i is the index of the pose. The user is free to choose how many elements the buffer contains, but for illustrative purposes, in this report it will be assumed to have eleven values, as shown in figure 2.

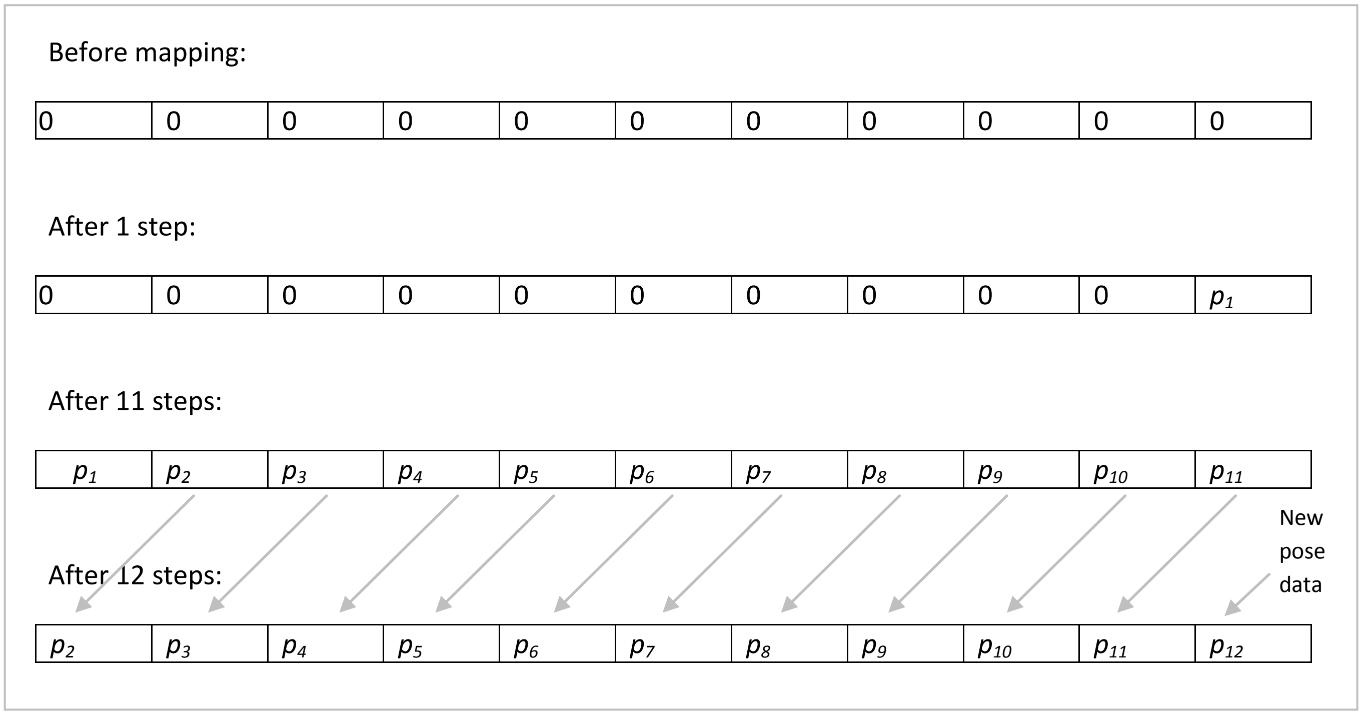
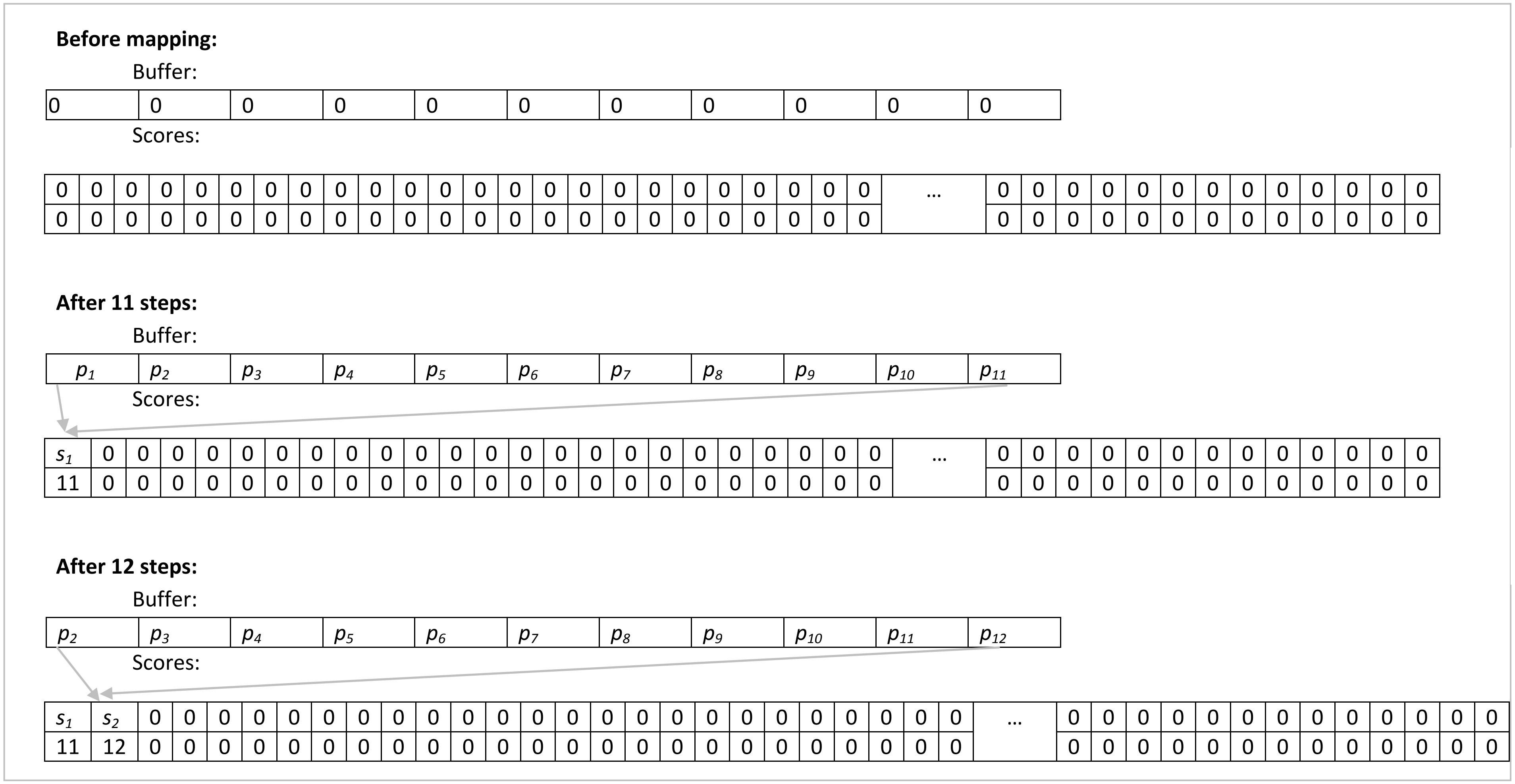

Before mapping:
0 0 0 0 0 0 0 0 0 0 0
After 1 step:
0 0 0 0 0 0 0 0 0 0 p_1
After 11 steps:
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11}
After 12 steps:
p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}
New pose data

Figure 2
Initialization and incrementing of buffer

In figure 2, each element p_i is a pose of the form (x, y, θ, t). Before the robot begins mapping, all elements are zero. As each new pose p_i is recorded, it is placed in the highest (eleventh, in this example) element of the buffer and the lowest (oldest) element of the buffer is dropped. The purpose of this buffer is to provide data for continual calculation of the straightness score matrix, which characterizes the local straightness of the trajectory.

The straightness score matrix S is used to calculate turning angles when turns in the trajectory are detected. Like the buffer data, the straightness score matrix could be a moving buffer, but for simplicity, in this example it is assumed to be a continually expanding matrix. Before the robot begins mapping, all elements of the straightness score matrix are zero. Once the robot has mapped eleven poses, the algorithm begins to calculate straightness scores. Each score is associated with a specific pose, and the score is the straightness of the set of poses consisting of that pose plus the ten prior poses.

The straightness score matrix has two rows. For each column j, row 1 holds a straightness score s_j, which is calculated from a set of eleven poses. Row 2 holds the index of the most recent of those eleven poses. New straightness score data is inserted in the higher elements of the matrix S. Figure 3 illustrates how elements of the straightness score matrix are associated with corresponding buffer vector elements.



Before mapping:

Buffer:

0	0	0	0	0	0	0	0	0	0	0

Scores:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0

After 11 steps:

Buffer:

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}

Scores:

s_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0

After 12 steps:

Buffer:

p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}

Scores:

s_1	s_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0	0	0	0
11	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3
Association of straightness score matrix with buffer data

The straightness score is calculated from the buffer values according to the procedure shown in figure 4.

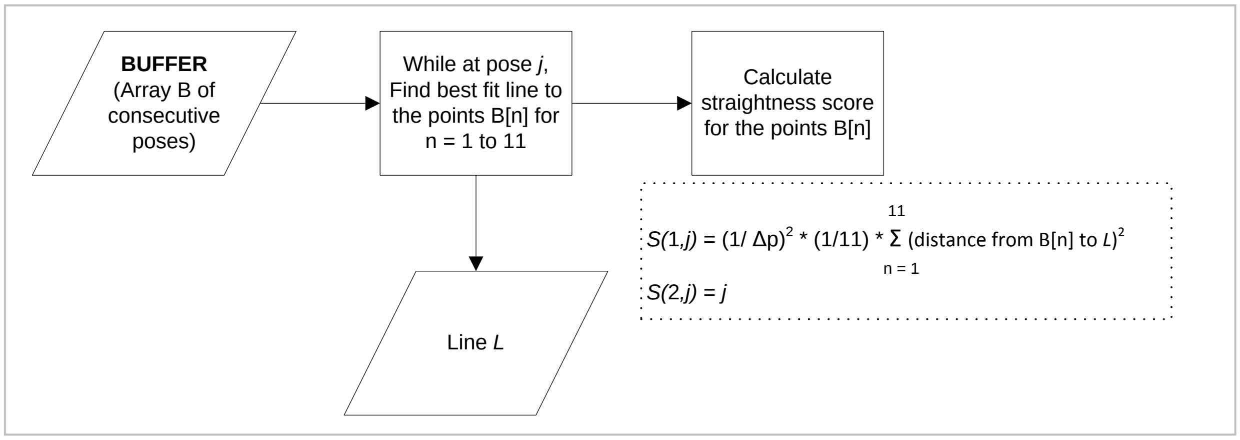

BUFFER
(Array B of consecutive poses)
While at pose *j*, Find best fit line to the points B[n] for n = 1 to 11
Calculate straightness score for the points B[n]
$S(1,j) = (1/\Delta p)^2 * (1/11) * \sum_{n=1}^{11} (\text{distance from } B[n] \text{ to } L)^2$
$S(2,j) = j$
Line *L*

Figure 4
Straightness score calculation

Let the buffer vector be denoted as $B(n)$, where n ranges from 1 to 11. The first task in calculating the straightness score is to find the best fit line L to the eleven points in the buffer; only the x and y elements of the poses are used in this calculation.

The next task is to calculate the corresponding values of the straightness score array, S. For a particular column index j, $S(1, j)$ is a straightness score and is derived from the set of eleven (x, y) points of pose data in the buffer, as illustrated in figure 3. $S(1, j)$ is defined as the normalized (to the robot's step distance Δp) mean squared distance from each of the eleven points to the best-fit line L.

$$S(1, j) = (1/\Delta p)^2 * (1/11) * \sum_{n=1}^{11} (\text{distance from } B[n] \text{ to } L)^2 \quad (1)$$

The second row element of S at column j is the pose index of the final buffer element that was used to calculate $S(1, j)$, as illustrated in figure 3. Thus,

$$S(2, j) = j + N_b - 1 \quad (2)$$

where N_b is the number of elements in the buffer.

Feature Descriptor and its Derivation for Trajectory Based Algorithm

The feature descriptor for the trajectory based algorithm is the data shown in table 1 about each detected corner in the trajectory.

Table 1
Descriptor values for trajectory based algorithm

Descriptor values for trajectory based algorithm	Variable name for descriptor value
Overall designation of *i*th corner descriptor	C_i
(*x, y*) coordinates of the vertex	$C_i.x$, $C_i.y$
Vertex angle	$C_i.\theta$
Angular orientation of each leg in a global reference frame	$C_i.\alpha L1$, $C_i.\alpha L2$
Time when the corner was traversed	$C_i.t$

In table 1, the notation is borrowed from concepts in object oriented programming. The corner descriptor C_i is conceptualized as an object whose members are the vertex coordinates x and y, the vertex angle θ, the traversal time t, as well as the angular orientations of the two legs, $\alpha L1$ and $\alpha L2$ connected to the vertex. The following is a description of how these pieces of information are derived.

As described previously, as the robot travels and each new pose is recorded, the buffer is updated with the new pose data and the time when the data was measured. Then, the straightness score is calculated using the current pose plus the ten immediately prior poses, as shown in equation 1. The new straightness score is then saved in the matrix S (fig. 3).

At this point, now that the straightness score matrix S has been updated with the latest data, prior values in the matrix are reviewed to determine whether a corner is present in the recent parts of the trajectory. The next task is to determine whether the matrix of straightness scores indicates the presence of a corner, and this is done as follows. The straightness score corresponding to a particular trajectory element is the deviation of the trajectory from the locally best fitted line to the past 11 trajectory elements. A corner would be recognizable as a local maximum in that deviation. Thus, we are looking for a local maximum in the elements of the straightness score matrix that correspond to the past several trajectory elements.

How far back in the trajectory we look for corners depends on the "size" of a corner in terms of the number of trajectory steps. In the example presented here, we assume that complete traversal of a corner can take as many as seven steps. Therefore, the next step in the algorithm is to examine the past seven values of the second row of S to see whether it has a maximum that exceeds a user-defined threshold value, *minStraightnessVal*. Recall that the score values of S in its first row are dimensionless – the deviation of the trajectory points from the local best-fit line is normalized to the step size in the trajectory.

A dimensionless threshold can be selected for use in evaluating the value of S in its first row, seven steps ago. The value of this threshold should be selected based on an experimental study of robot trajectories, while traversing corners of various angles. The threshold should be high enough to filter out gentle curves in the trajectory, e.g., those due to the robot's meandering, but it should also be low enough to enable recognition of genuine corners.

In the corner detection procedure, three questions (to be specified shortly) are asked in order to determine whether a corner should be defined in connection with the straightness score $S(1, m)$, where m is the index of the current trajectory element of the robot. Since we are looking at the past seven trajectory elements, we need to examine elements $S(1, m-6)$ through $S(1, m)$ of matrix S to determine whether there is a clearly defined local maximum among these values. In particular, we examine the middle of these seven elements to determine whether it is a local maximum. Since the set of elements of S are examined in a sliding window manner, all elements of S will end up being examined in this method.

The middle of the seven elements is *S(1, m-3)*. This element is examined according to the following three tests to determine whether it is associated with a corner.

First, *S(1, m-3)* is compared to its neighbors in the straightness score matrix according to these two criteria

(a) the value of the score *S(1, m-3)* is greater than or equal to the score of each of its closest neighbors, *S(1, m-4)* and *S(1, m-2)*, **and**

(b) *S(1, m-3)* is greater than **all** of these more distant neighbors: *S(1, m-6)*, *S(1, m-5)*, *S(1, m-1)*, *S(1, m)*.

Second, if *S(1, m-3)* is a local maximum according to the first test, it is compared to the threshold value to determine whether it is a sharp enough maximum. If *S(1, m-3)* is above threshold, and is also greater than its neighbors as described, then a corner detection has occurred.

This corner, together with its data shown in table 1, will be cataloged in a "Junction Database," as described in the next section. The only situation where this corner would not be cataloged is when it had been previously cataloged during this excursion of the robot. Thus, the third test is whether this corner has already been cataloged.

The corner detection scheme described previously is illustrated in figure 5.

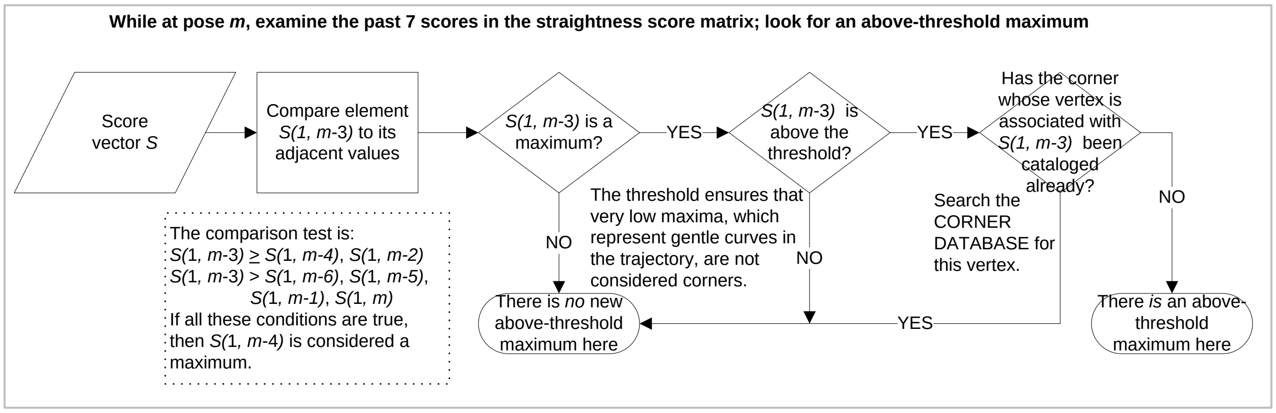

While at pose *m*, examine the past 7 scores in the straightness score matrix; look for an above-threshold maximum
Score vector *S*
Compare element *S(1, m-3)* to its adjacent values
S(1, m-3) is a maximum?
YES
S(1, m-3) is above the threshold?
YES
Has the corner whose vertex is associated with *S(1, m-3)* been cataloged already?
NO
NO
The threshold ensures that very low maxima, which represent gentle curves in the trajectory, are not considered corners.
NO
Search the CORNER DATABASE for this vertex.
YES
There is *no* new above-threshold maximum here
There *is* an above-threshold maximum here
The comparison test is:
S(1, *m*-3) ≥ *S*(1, *m*-4), *S*(1, *m*-2)
S(1, *m*-3) > *S*(1, *m*-6), *S*(1, *m*-5), *S*(1, *m*-1), *S*(1, *m*)
If all these conditions are true, then *S*(1, *m*-4) is considered a maximum.

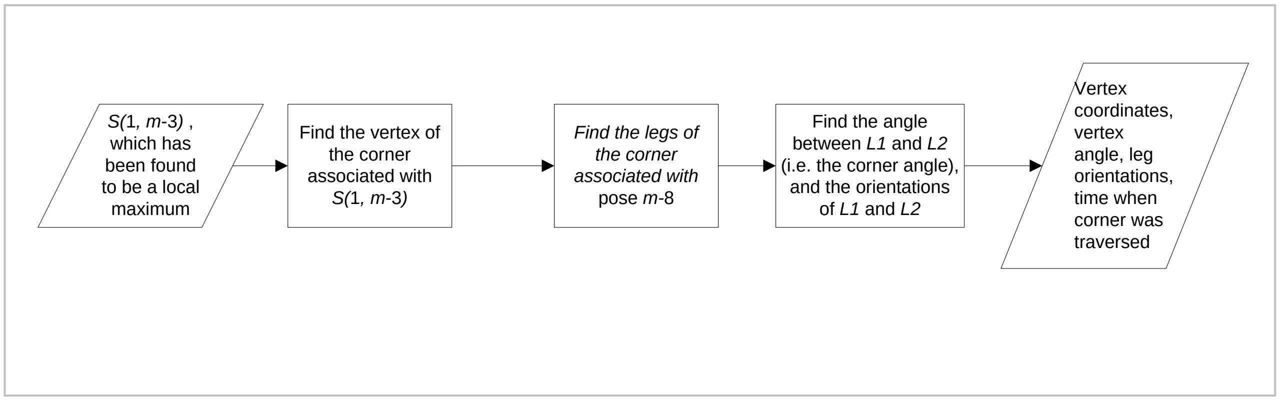
Figure 5
Corner detection procedure for trajectory based algorithm

There are two possible outcomes of the corner detection process shown in figure 5: either there is or is not an above-threshold maximum at *S(1, m-3)*. If there is not such a maximum, then no further processing is done at this step, and the robot travels to the next step in its trajectory. If there is such a maximum, the next task is to derive the descriptor data shown in table 1.

The first piece of data to be calculated is the vertex of the corner. We know that *S(1, m-3)* is a local, above-threshold maximum. This straightness score is derived from the past eleven trajectory values that are in the buffer, i.e., *m-13* through *m-3*. We shall define the vertex of the corner as being the **middle** trajectory element of this set, i.e., *m-8*.

The next task is to determine coordinates for the legs of the corner. The legs are defined as the best fit line segment for two sets of trajectory elements – those traversed right before encountering the vertex and those traversed right afterward. That is, one leg, called L1, is the best fit line segment for trajectory elements *m-13* through the vertex, *m-8*. The other leg, called L2, is the best fit line segment for the vertex, trajectory element m-8 through the trajectory element *m-3*. Once the coordinates of the line segments L1 and L2 are determined, the angle between them can be found from trigonometry. Although the angle is not an independent number from the data for L1 and L2, it is convenient to retain it as part of the corner descriptor since it is used later in the algorithm to compare two descriptors.

The process for calculating the data for a corner descriptor is summarized in figure 6.


S(1, *m*-3) , which has been found to be a local maximum
Find the vertex of the corner associated with *S*(1, *m*-3)
Find the legs of the corner associated with pose *m*-8
Find the angle between *L1* and *L2* (i.e. the corner angle), and the orientations of *L1* and *L2*
Vertex coordinates, vertex angle, leg orientations, time when corner was traversed

.

Figure 6
Descriptor data calculation procedure

At this point in the algorithm, all the data for the descriptor corresponding to the current pose in the trajectory (table 1) has been calculated. As each new descriptor is calculated for turns made in the trajectory, the new descriptor is cataloged in a Junction Database. This database is a repository of all previously made turns and is used in comparing newly made turns with previously made turns to determine whether place recognition has occurred.

Descriptor Comparison Process for Trajectory Based Algorithm

The next task is to compare the newly calculated descriptor with descriptors from previous poses in the trajectory. For long trajectories, it can be computationally intensive to compare the new descriptor with every descriptor that was ever calculated. Therefore, it would be useful to define a search region, i.e., a subset of previously calculated descriptors to use for descriptor comparisons. This subset would consist of the set of cataloged descriptors that are most likely to match the current descriptor. To define some terminology that will simplify the discussion, the currently measured descriptor will be called the “target descriptor” C^T, and the descriptor in the Junction Database with which the target descriptor is being compared will be called the “candidate descriptor” C^C.

The following criteria are used to select which descriptors in the Junction Database are most likely to match the current descriptor and will therefore be compared in detail with the current descriptor

$$\text{Criterion a: } |C^T.t - C^C.t| > t_{min} \tag{3}$$

$$\text{Criterion b: } |C^T.t - C^C.t| < t_{max} \tag{4}$$

$$\text{Criterion c: } (\,(C^T.x - C^C.x)^2 + (C^T.y - C^C.y)^2\,)^{1/2} > d_{min} \tag{5}$$

$$\text{Criterion d: } (\,(C^T.x - C^C.x)^2 + (C^T.y - C^C.y)^2\,)^{1/2} < d_{max} \tag{6}$$

Criterion (a) states that the difference in traversal times between the target and candidate descriptors is **greater than** a user-defined threshold, t_{min}. It is possible that the robot will meander a short distance from a certain location and then backtrack to that same location. We would not want to consider these two visits to that location as a place recognition event. Therefore, criterion (a) ensures that we consider only revisits to a certain place that occur at least t_{min} earlier than the time of the current visit.

Criterion (b) states that the difference in traversal times between the target and candidate descriptors is **less than** a user defined threshold, t_{max}. The reason for criterion (b) is that we want to limit the number of candidates considered in order to reduce the computation time. Therefore, if a candidate was traversed a very long time ago, we do not evaluate it for place recognition.

Criterion (c) states that the distance between the target corner and the candidate corner is **greater than** a user-defined threshold d_{min}. The intent of this criterion, similar to that of criterion (a), is to filter out meanderings.

Criterion (d) states that the distance between the target corner and the candidate corner is **less than** a user-defined threshold d_{max}. The intent of this criterion, similar to that of criterion (b), is to filter out candidate corners that are very distant from the target corner in order to reduce computation time.

Now that the four criteria for the search region are defined, the Junction Database is examined to determine which, if any, candidate descriptors in the search region satisfy all four criteria. If no candidates satisfy all four criteria, then no further calculations are done at this step in the trajectory, and the robot travels further.

However, if some candidates that do satisfy all four criteria are found, then these candidate descriptors are examined in order to determine how closely they match the target descriptor. Three similarity tests are performed on the candidates to determine their similarity to the target descriptor (1) proximity of vertices, (2) similarity of corner angles, and (3) similarity of corner orientations. For each of these tests, a metric is calculated, and then a matching score is calculated as a weighted sum of the three metric values. The metrics and matching score are defined in such a way that the lower the value, the better the match. The three metrics are defined as

$$proximityMetric = \left((C^T.x - C^C.x)^2 + (C^T.y - C^C.y)^2 \right)^{½} / \Delta p \quad (7)$$

$$cornerAngleMetric = (C^T.\vartheta - C^C.\vartheta) / 360 \text{ degrees} \quad (8)$$

$$cornerOrientationMetric = \left(½\,|C^T.\alpha L1 + C^T.\alpha L2| - ½\,|C^C.\alpha L1 + C^C.\alpha L2| \right) / 360 \text{ degrees} \quad (9)$$

Each of the three metrics is normalized so that it is a dimensionless number. The proximity metric is the Euclidean distance between the vertices of the target and candidate corners, normalized to the step size in the trajectory, Δp. The corner angle metric is the difference between corner angles of the target and candidate corners, normalized to 360 deg. The corner orientation metric is the difference between the centerline orientations for the target and candidate corners, normalized to 360 deg.

The next part of the algorithm calculates the matching score for each candidate and selects the candidate with the lowest score. This algorithm assigns greater weights to the proximity metric and the corner angle metric than it assigns to the corner orientation metric. That is because drift errors in the trajectory calculation tend to cause large-scale errors in the map, in which the angle between the target corners's legs changes very little, but the entire target corner is rotated in a large-scale sense. The corner orientation metric is more strongly affected by this type of error than are the other metrics. Therefore, if the corner orientation metric is too heavily weighted, then false negative judgments would result, in which two corners that actually represent the same location are judged to be from different locations.

Based on these ideas, the suggested weighting to the proximity, corner angle, and corner orientation metrics are 0.4, 0.4, and 0.2 respectively. Thus, the definition of the matching score is

$$\begin{aligned} matchingScore = \quad & 0.4 * proximityMetric + \\ & 0.4 * cornerAngleMetric + \\ & 0.2 * cornerOrientationMetric \end{aligned} \tag{10}$$

These weights could, of course, be changed if the situation so warrants. Figure 7 shows the procedure for scoring the match between the target and candidate corners. The output of this procedure is the candidate corner that best matches the target corner and the score that tells how well it matches the target corner.

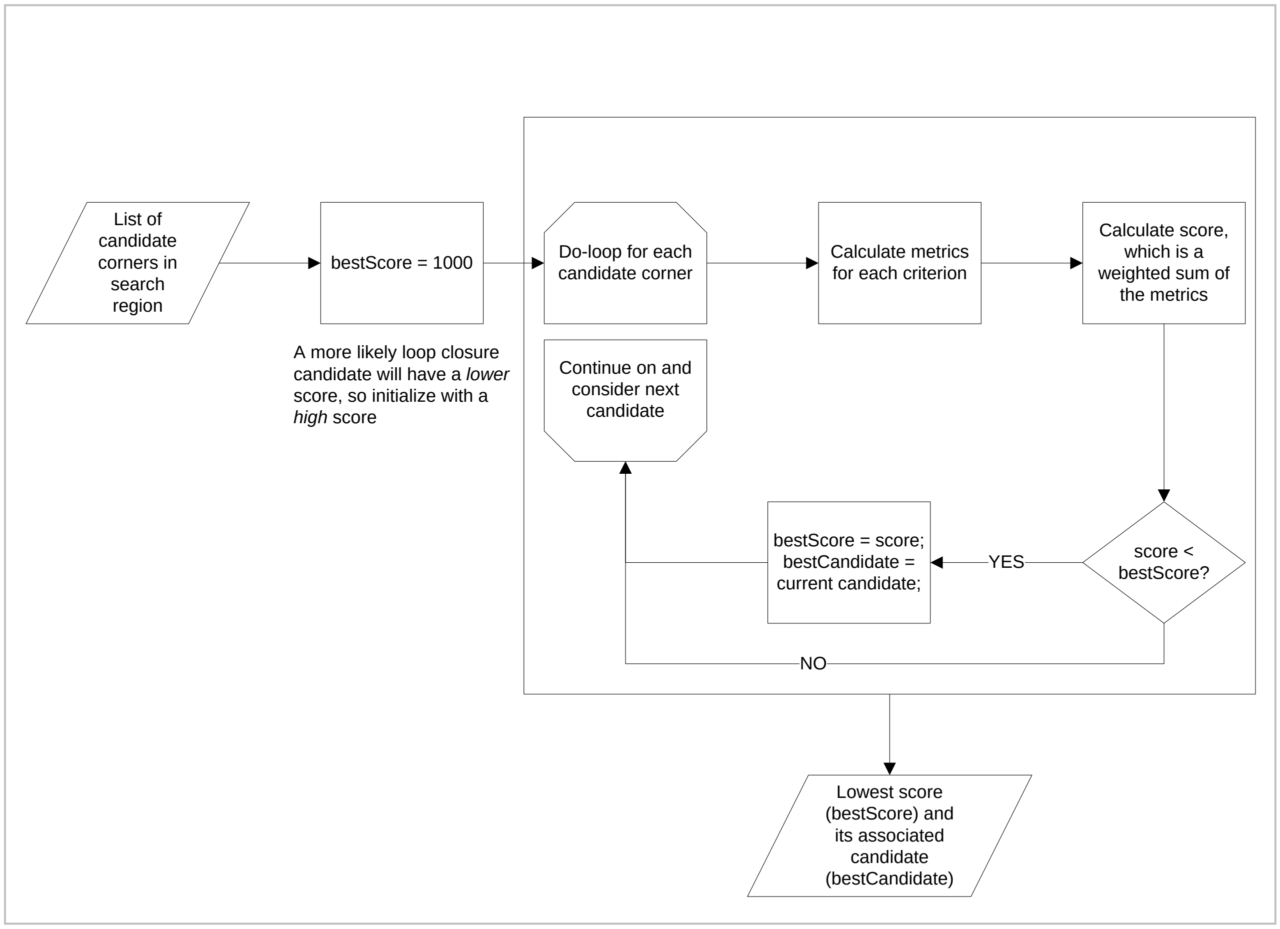

List of candidate corners in search region
bestScore = 1000
A more likely loop closure candidate will have a *lower* score, so initialize with a *high* score
Do-loop for each candidate corner
Calculate metrics for each criterion
Calculate score, which is a weighted sum of the metrics
score < bestScore?
YES
bestScore = score; bestCandidate = current candidate;
NO
Continue on and consider next candidate
Lowest score (bestScore) and its associated candidate (bestCandidate)

Figure 7
Scoring method for evaluation of target and candidate match for trajectory based algorithm

Once this lowest score is calculated, it is compared with a user-specified threshold value s_{max}, which is the largest acceptable score where the match between the target and candidate corners is considered a place recognition. If this lowest score is greater than s_{max}, then there is no viable place recognition candidate, so no further calculations are done at this step in the trajectory, and the robot travels further.

Derivation of Output Data for Trajectory Based Algorithm

If this point in the algorithm is reached, then the best matched candidate is considered a recognized place, and the output data that is to be sent to the trajectory correcting algorithm has to be calculated. For the trajectory based algorithm, part of the output data is the two alignment points, namely, the vertices of the target and best-candidate corners

$$\text{Output corner vertices} = \{\mathrm{C^T.x}, \mathrm{C^T.y}, \mathrm{C^C.x}, \mathrm{C^C.y}\} \tag{11}$$

The remainder of the output data is the matching score associated with this match. In effect, the robot "tells" the trajectory adjustment program, "My calculations show me that I am at the target corner, but I think I am at the candidate corner, so please adjust the overall trajectory to co-locate the two corners. My confidence in this match is indicated by the matching score." The complete set of output data for the trajectory based algorithm is shown in table 2.

Table 2
Output data for trajectory based algorithm

Target descriptor, C^T
Candidate descriptor, C^C
Matching score, *matchingScore*

Overall Summary of Trajectory Based Algorithm

Table 3 summarizes the user selectable parameters for the trajectory based algorithm.

Table 3
List of selectable parameters for trajectory based algorithm

Parameter description	Variable name	Assumed value, if any
Robot's step distance	Δp	
Number of elements in buffer		11
Straightness score matrix number of columns		Indefinitely large
Number of steps in traversal of a corner		7
Threshold straightness for identifying corners	*minStraightnessVal*	
Minimum time for search region	t_{min}	
Maximum time for search region	t_{max}	
Minimum distance for search region	d_{min}	
Maximum distance for search region	d_{max}	
Threshold score for place recognition candidates	s_{max}	

Figure 8 shows a high level overview of the entire trajectory based algorithm. An advantage of this algorithm is that it does not depend on the wall structure, other than the fact that it assumes that the trajectory is constrained by some type of wall structure. This algorithm would not be effective at all in an unconstrained environment because, in such an environment, a turn in the trajectory has no connection with any particular place.

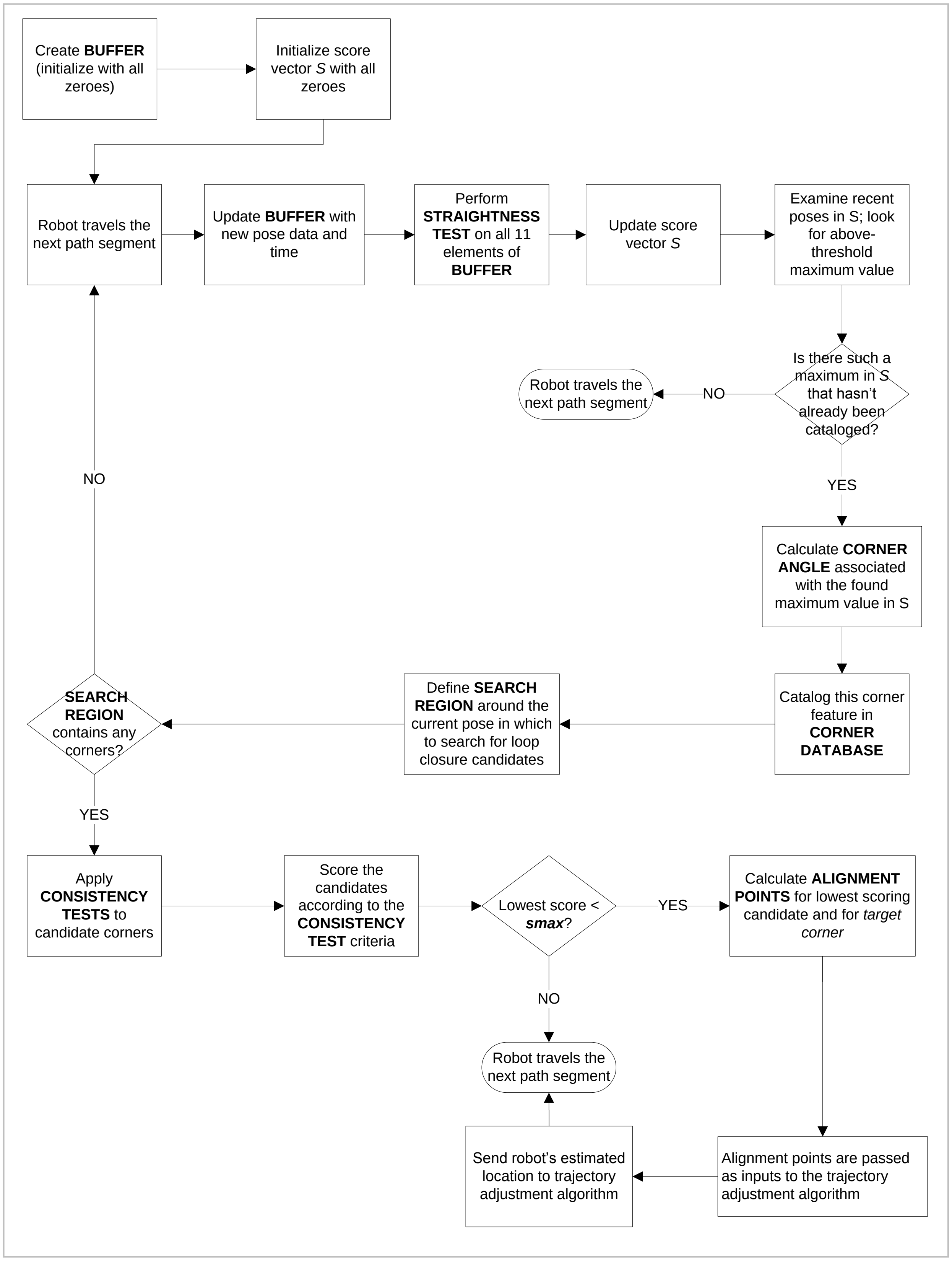

Create **BUFFER** (initialize with all zeroes)
Initialize score vector *S* with all zeroes
Robot travels the next path segment
Update **BUFFER** with new pose data and time
Perform **STRAIGHTNESS TEST** on all 11 elements of **BUFFER**
Update score vector *S*
Examine recent poses in S; look for above-threshold maximum value
Is there such a maximum in *S* that hasn't already been cataloged?
NO
Robot travels the next path segment
YES
Calculate **CORNER ANGLE** associated with the found maximum value in S
Catalog this corner feature in **CORNER DATABASE**
Define **SEARCH REGION** around the current pose in which to search for loop closure candidates
SEARCH REGION contains any corners?
NO
YES
Apply **CONSISTENCY TESTS** to candidate corners
Score the candidates according to the **CONSISTENCY TEST** criteria
Lowest score < ***smax***?
YES
Calculate **ALIGNMENT POINTS** for lowest scoring candidate and for *target corner*
NO
Robot travels the next path segment
Alignment points are passed as inputs to the trajectory adjustment algorithm
Send robot's estimated location to trajectory adjustment algorithm

Figure 8
High level flow chart for trajectory based algorithm

LT-Junction Based Algorithm

This section describes, for the LT-junction based algorithm, the input data and its preprocessing, the feature descriptor and its derivation from the preprocessed input data, the descriptor comparison process, and the derivation of output data.

The LT-junction based algorithm is useful for place recognition and loop closure detection in situations in which a robot is driving in a network of featureless passageways with nominally flat, parallel walls that constrains its motion to either nominally straight line driving or turning from one path segment onto another. It uses hallway junction characteristics to label and compare visited places, and to search for recognized places.

This algorithm detects junctions shaped like an “L” or like a “T,” though the intersections do not need to have right angles as the alphabet letters have. Like the trajectory based algorithm, this algorithm maintains a sliding buffer of eleven consecutive poses to enable detection of junction features. This algorithm explicitly makes use of wall features; therefore, wall data, as well as trajectory data is collected at each step of the trajectory. In particular, the buffer contains pose information for each entry (as was the case for the trajectory based algorithm), and it also contains data about the local wall segment endpoints and orientations, as well as the distance from the robot to each wall.

The algorithm catalogs each junction it detects into a database and compares newly detected junctions to the junctions in the database in order to determine whether a place recognition event has occurred.

Input Data and Preprocessing for LT-Junction Based Algorithm

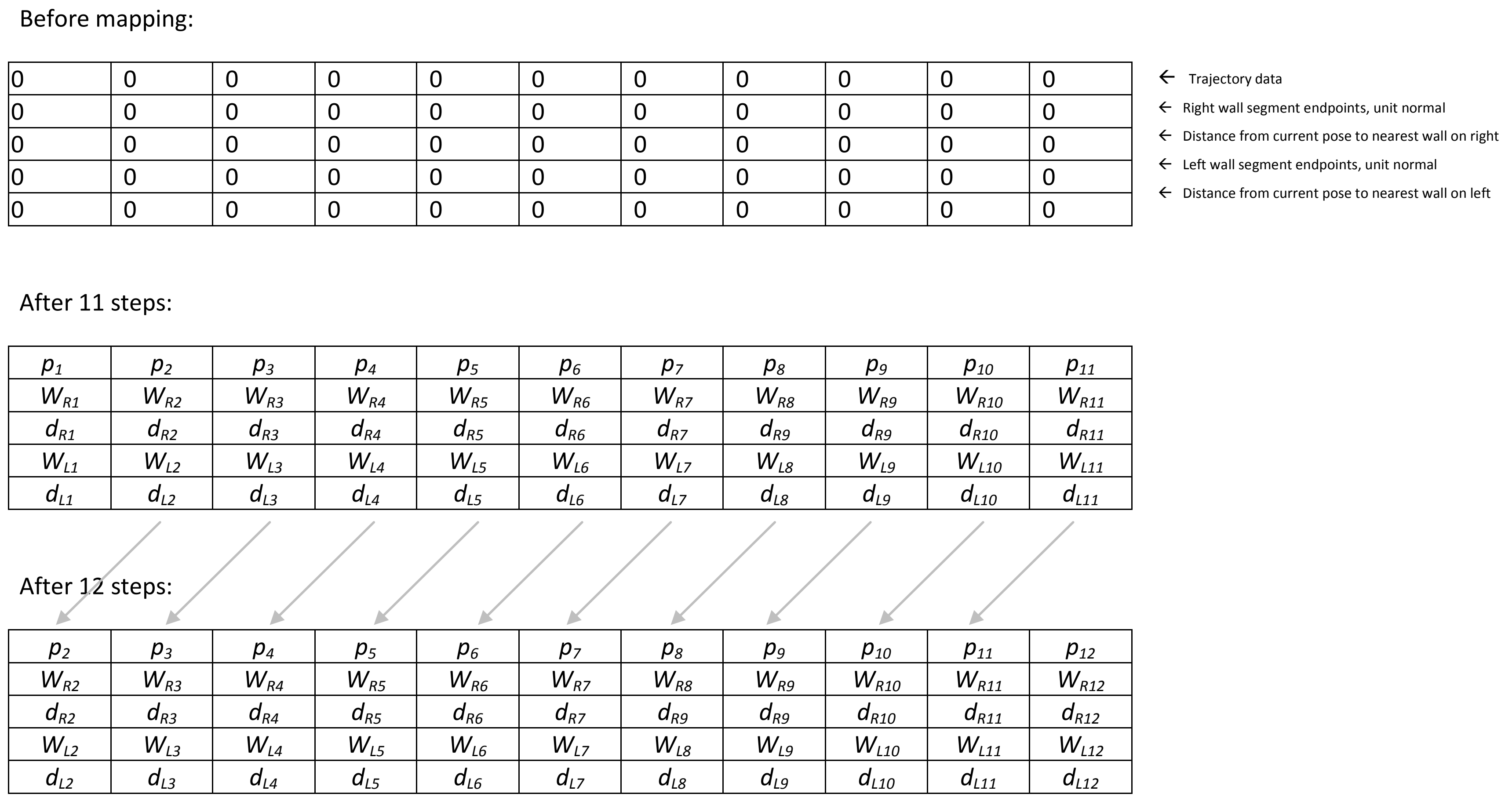
The input data for the LT-junction based algorithm consists of two types of data, robot pose data and data about the adjacent walls. The pose data is a time stamped sequence of 2D robot poses that could be measured by inertial and possibly other sensors. Each pose is of the form (x, y, θ, t), where x and y are the 2D coordinates of the center of the robot projected onto the ground in a global frame of reference, θ is orientation angle of the robot, and t is the time at which the pose data was measured. As the robot drives along, the pose is measured and recorded at regular intervals, resulting in a continuous stream of pose data.

At each pose, data about the wall sections on each side of the robot are measured. One type of wall data is the line segment which is the projection – onto the x-y (floor) plane – of the wall section that is within the field of view of the robot’s sensors at the current pose. The line segment is expressed as a set of $(x_i, y_i, x_f, y_f, \mathrm{t})$ points where t is the time of the pose at which that wall section was measured. Here x_i and y_i are the coordinates of the initial point in the segment, and x_f and y_f are the coordinates of the final point. Another type of wall data is the unit normal vector to each segment in the x-y plane. Although this unit normal vector is entirely dependent on the wall segment data, it is convenient to retain the normal vector data for use when comparing two junction descriptors. Yet another type of wall data recorded in the buffer is the distance from the current pose to the nearest wall on each side of the robot. All of the wall data can be derived from point cloud data from a LIDAR system. The wall data can be obtained by using a plane-based segmentation algorithm on the point cloud to extract side walls and then projecting the side walls down onto the x-y plane.

A moving buffer is used to store the most recently recorded set of consecutive pose values and data about the adjacent wall segments. The user is free to choose how many pose values the buffer contains, but for illustrative purposes, in this report it will be assumed to have eleven values, and therefore, the buffer will have eleven columns, such that each column contains pose data and wall data measured from that pose. Using this number of columns enables collection of enough data in order to derive local properties of the wall structure when looking for junctions.

The buffer has five rows: the top row is the set of consecutive poses of the trajectory that are compactly denoted by p_i for the ith trajectory step. The second row is wall segment data of points in the nearest right wall segment, denoted as W_{Ri} for the ith trajectory step. The third row is the wall distance to nearest right wall segment, denoted as d_{Ri}. The fourth row is the wall segment data of points in the nearest left wall segment, denoted as W_{Li}. The fifth row is the wall distance to nearest left wall segment, denoted as d_{Li}.

Buffer updating works as follows. As data for each new step in the trajectory is recorded and/or calculated, the oldest data – the data in the lowest-index column – is discarded, each other column of data is shifted to a lower index, and the newest data is inserted into the highest-index column. Figure 9 illustrates the structure and updating of the buffer for the LT-junction based algorithm. Figure 10 shows the process for recording and updating the buffer data.



Before mapping:

0	0	0	0	0	0	0	0	0	0	0	← Trajectory data
0	0	0	0	0	0	0	0	0	0	0	← Right wall segment endpoints, unit normal
0	0	0	0	0	0	0	0	0	0	0	← Distance from current pose to nearest wall on right
0	0	0	0	0	0	0	0	0	0	0	← Left wall segment endpoints, unit normal
0	0	0	0	0	0	0	0	0	0	0	← Distance from current pose to nearest wall on left

After 11 steps:

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
W_{R1}	W_{R2}	W_{R3}	W_{R4}	W_{R5}	W_{R6}	W_{R7}	W_{R8}	W_{R9}	W_{R10}	W_{R11}
d_{R1}	d_{R2}	d_{R3}	d_{R4}	d_{R5}	d_{R6}	d_{R7}	d_{R9}	d_{R9}	d_{R10}	d_{R11}
W_{L1}	W_{L2}	W_{L3}	W_{L4}	W_{L5}	W_{L6}	W_{L7}	W_{L8}	W_{L9}	W_{L10}	W_{L11}
d_{L1}	d_{L2}	d_{L3}	d_{L4}	d_{L5}	d_{L6}	d_{L7}	d_{L8}	d_{L9}	d_{L10}	d_{L11}

After 12 steps:

p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
W_{R2}	W_{R3}	W_{R4}	W_{R5}	W_{R6}	W_{R7}	W_{R8}	W_{R9}	W_{R10}	W_{R11}	W_{R12}
d_{R2}	d_{R3}	d_{R4}	d_{R5}	d_{R6}	d_{R7}	d_{R9}	d_{R9}	d_{R10}	d_{R11}	d_{R12}
W_{L2}	W_{L3}	W_{L4}	W_{L5}	W_{L6}	W_{L7}	W_{L8}	W_{L9}	W_{L10}	W_{L11}	W_{L12}
d_{L2}	d_{L3}	d_{L4}	d_{L5}	d_{L6}	d_{L7}	d_{L8}	d_{L9}	d_{L10}	d_{L11}	d_{L12}

Figure 9
Initialization and updating of buffer matrix using wall data

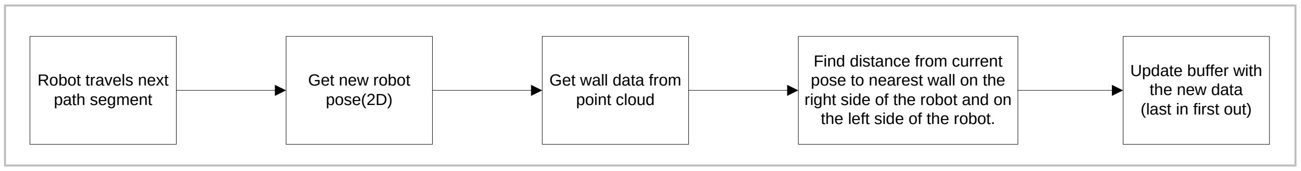

Robot travels next path segment
Get new robot pose(2D)
Get wall data from point cloud
Find distance from current pose to nearest wall on the right side of the robot and on the left side of the robot.
Update buffer with the new data (last in first out)

Figure 10
Process for updating buffer data

Feature Descriptor and its Derivation for LT-Junction Based Algorithm

The algorithm first determines whether one or both walls have gaps in them by examining buffer elements. Such gaps indicate the presence of doorways, hallways, and/or rooms.

Wall gaps are detected by searching prior buffer data (first data for the right side wall and then data for the left side wall[1]) for maxima in the nearest-wall-distance versus trajectory-pose function. Such maxima (if they are above threshold) indicate the presence of a T-junction. It is important to first search one wall and then the other, rather than both of the simultaneously, in order to avoid the ambiguity of detecting a + shaped junction. In this algorithm, a + shaped junction is represented as a pair of back-to-back T-junctions.

In order to better understand why a maximum in the nearest wall distance versus trajectory pose function indicates the presence of a wall gap, it is instructive to consider in detail two types of T-junctions: straight-T junctions and bent-T junctions. A straight-T junction is one in which the robot travels straight past a not taken path, as shown in figure 11. A bent-T junction is one in which the robot turns past a not taken path, as shown in figure 12.

[1] At this point, it is important to keep track of which side of the buffer data we are considering, i.e., data for the right side wall or for the left side wall. This information will be important later when we calculate the descriptor parameters.

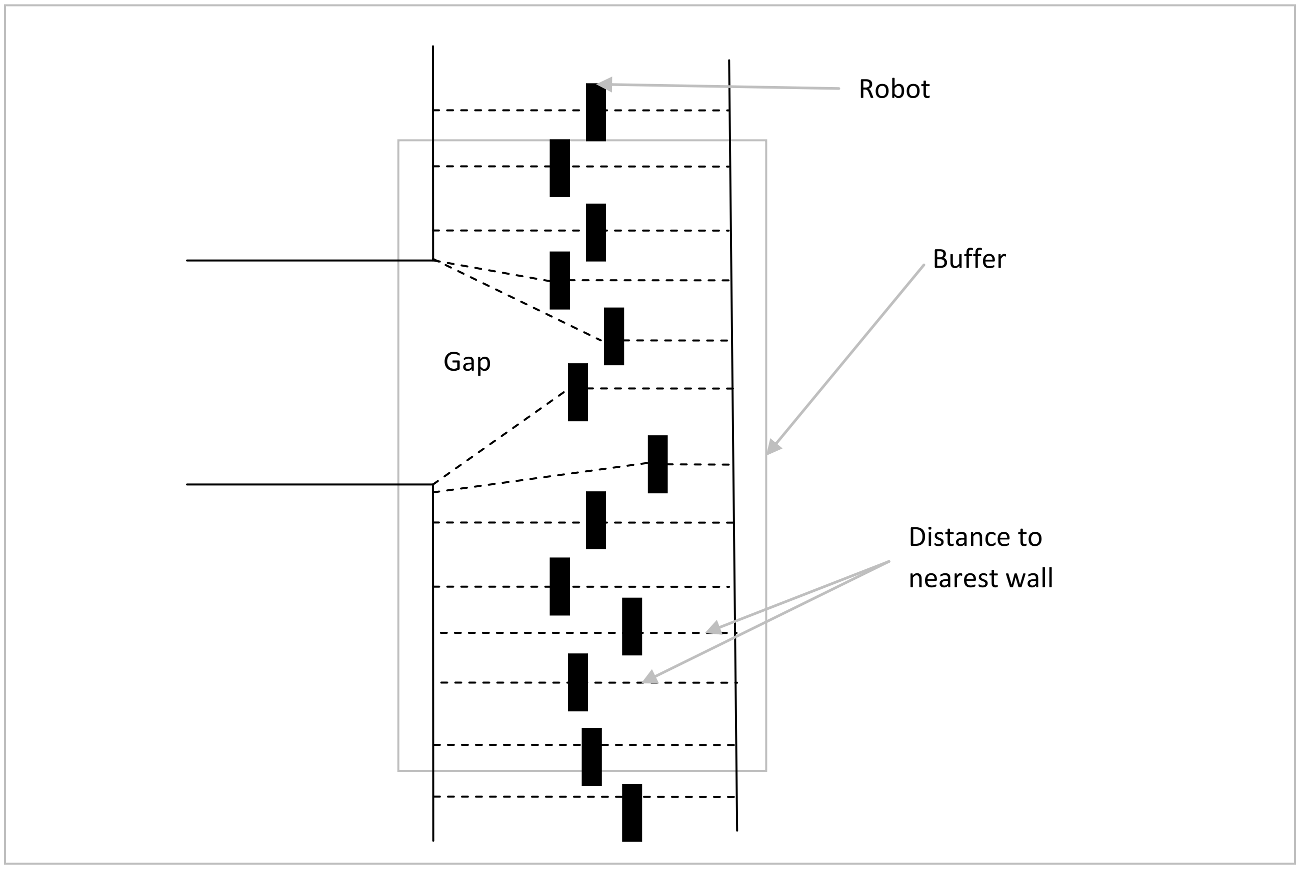


Figure 11
Straight T-junction

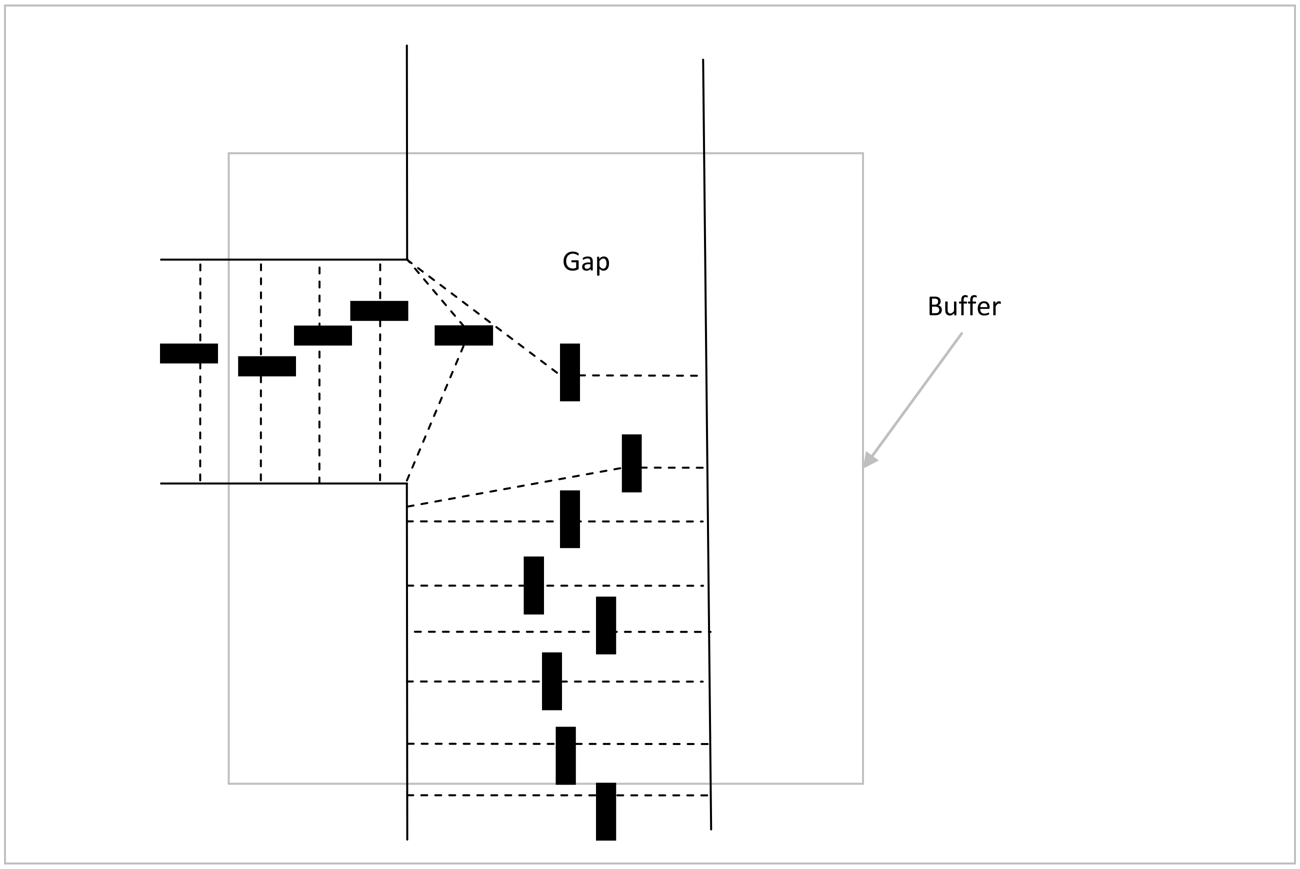


Figure 12
Bent-T junction

In figures 11 and 12, each solid rectangle represents the position of the robot at a point in its trajectory. The dashed lines on each side of a solid rectangle represent the distance from the "origin" of the robot to the nearest wall on its right and left side. Here, the "origin" of the robot is defined as the center point of the robot projected down onto the ground plane. In both of these figures, the poses whose data is currently in the buffer are enclosed in a box labeled "buffer." Both figures illustrate some meandering in the robot's trajectory since in practice the robot often does meander somewhat as it drives.

Figures 11 and 12 illustrate the calculation of the nearest-wall-distance versus trajectory-pose function. In particular, **three functions** of the robot position are examined to detect a gap

- The distance of the "origin of the robot" to the nearest wall on its right
- The distance of the "origin of the robot" to the nearest wall on its left and
- The sum of these two distances

As these diagrams suggest, at locations where the robot drives along a straight hallway, the sum of the two distance functions is constant. However, at locations where the robot drives past, or turns onto, a wall gap, the sum of the two distance functions exhibits a maximum. Thus, a maximum in the sum function indicates the presence of a gap. While the robot is traversing the gap, the sum function could take on a range of values, depending on the robot's pose while in that region, and thus the sum function might not remain at its maximum value throughout the robot's traversal of the gap. However, before entering the gap region, and after leaving it, the robot is in straight walled regions in which the sum function is constant.

A gap is a region comprised of two sections of straight path in which the sum function is constant for some distance, with a section in between these two sections that has an above-threshold maximum value. More specifically, the maximum must have all of these **three criteria:**

1) At least three consecutive poses in the buffer have a constant sum function value equal to the hallway width

2) These poses are followed immediately by a maximum value (compared to the three prior and later poses) that is larger than the hallway width by a user-selectable margin of *deltaWidth*

3) Values in the buffer immediately following the maximum include at least three consecutive poses with constant sum function value equal to the hallway width

The process of deciding whether a sequence of poses contains a gap is illustrated in figure 13, which shows a set of several consecutive columns of the buffer. Definitions of the data in the rows of the buffer are shown in figure 9.

p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
W_{R2}	W_{R3}	W_{R4}	W_{R5}	W_{R6}	W_{R7}	W_{R8}	W_{R9}	W_{R10}	W_{R11}	W_{R12}
d_{R2}	d_{R3}	d_{R4}	d_{R5}	d_{R6}	d_{R7}	d_{R9}	d_{R9}	d_{R10}	d_{R11}	d_{R12}
W_{L2}	W_{L3}	W_{L4}	W_{L5}	W_{L6}	W_{L7}	W_{L8}	W_{L9}	W_{L10}	W_{L11}	W_{L12}
d_{L2}	d_{L3}	d_{L4}	d_{L5}	d_{L6}	d_{L7}	d_{L8}	d_{L9}	d_{L10}	d_{L11}	d_{L12}

Figure 13
Identification of wall distance maximum in buffer

For illustrative purposes, the current pose is the tenth pose[2], p_{10}. The seven poses of p_4 through p_{10} are to be considered. The middle pose of this set, p_7, is examined in order to determine whether it is a maximum. For p_7 to be a maximum, the three criteria would be:

(1) The sum function for each of p_4, p_5, and p_6 is equal to the hallway width

(2) The sum function for p_7 is larger than the hallway width by a margin of *deltaWidth*

(3) The sum function for each of p_8, p_9, and p_{10} is equal to the hallway width

Figure 14 shows the process for examining the buffer to determine whether a hallway segment has a gap in it.

[2] The data for p_{11} and p_{12} is shown in light gray because in our example, it has not yet been measured.

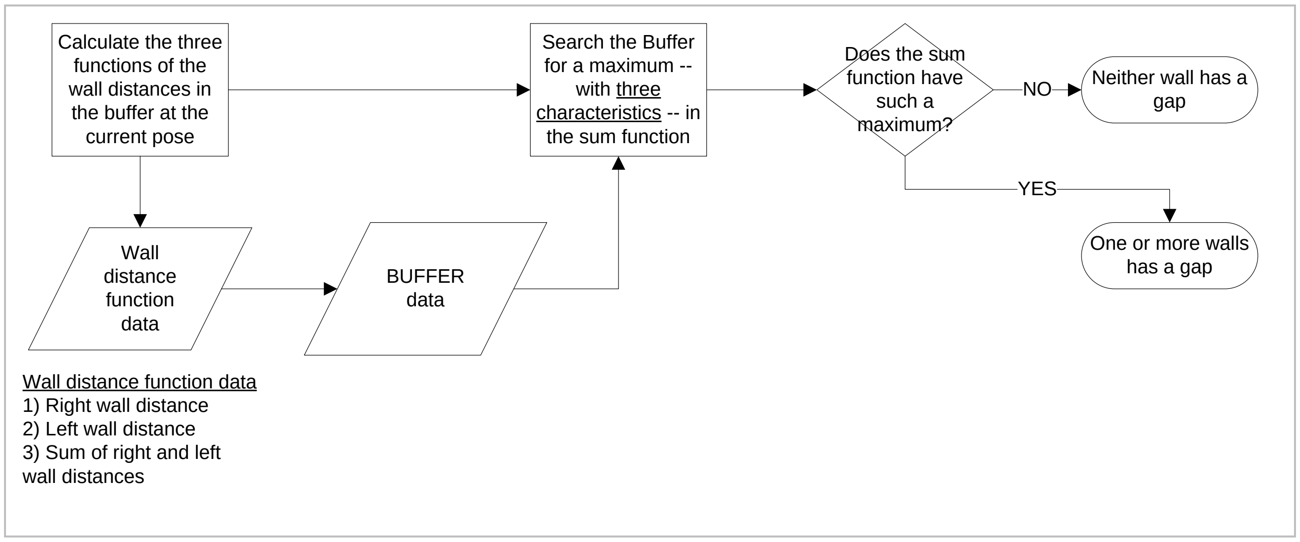

Calculate the three functions of the wall distances in the buffer at the current pose
Search the Buffer for a maximum -- with three characteristics -- in the sum function
Does the sum function have such a maximum?
NO
Neither wall has a gap
YES
One or more walls has a gap
Wall distance function data
BUFFER data
Wall distance function data
1) Right wall distance
2) Left wall distance
3) Sum of right and left wall distances

Figure 14
Process for determining whether a hallway has a gap

At this point in the algorithm, a significant decision must be made that determines what kind of descriptor is to be calculated for the section of the trajectory that is currently traversed. The decision is whether there is a gap in one or more walls, and the answer is obtained using the process shown in figure 14. If a gap has been detected, then the wall gap data will be calculated to obtain the T-junction descriptor. If no gap has been detected, then an L-junction is sought. If an L-junction is found, then corner data is calculated to obtain the L-junction descriptor. If neither a T-junction nor an L-junction is found, then no descriptor is calculated at this point in the path. The following two sections present the derivation of the T-junction and L-junction descriptor, respectively.

T-Junction Descriptor Derivation

The section of the algorithm that deals with derivation of the T-junction descriptor is arrived at if and only if a gap has been detected in the right or left wall, as determined by evaluating the three characteristics specified in the previous section.

The seven consecutive poses comprised of the current pose plus the previous six poses have been found to contain a maximum that indicates a wall gap. This set of poses is now examined in order to find parameters of the T-junction descriptor. With reference to figure 13, the set of poses p_4, p_5, and p_6 is referred to as the “BEFORE” region, the pose p_7 is referred to as the “DURING” region, and the set of poses p_8, p_9, and p_{10} is referred to as the “AFTER” region.

The next step is to find which poses correspond to the endpoints of the gap. The endpoints are designated as the last pose of the “BEFORE” region and the first pose of the “AFTER” region. In our example, the gap endpoints would be p_6 and p_8. This pose information will be used to look up the corresponding wall information in the buffer to more precisely determine the endpoints and location of the gap.

The buffer contains information about the walls on both sides of the robot – the right wall as well as the left wall. So, first verification is needed as to which side of the robot the gap is, and then, it can be determined which data to retrieve from the buffer. When the search of the buffer for maxima began, it started with the examination of the right side wall data, and then the left side wall data as described earlier. Using this information allows the side of the robot on which the identified gap exists to be determined.

Suppose, for illustrative purposes, that a gap has been found on the right side of the robot. We now retrieve the wall data for the gap endpoints, i.e., the wall data for poses p_6, p_7, and p_8, which is W_{R6}, W_{R7}, and W_{R8}. The wall data stored in these variables is the endpoints of the wall segment and the normal vector (in the floor plane) to the wall segment. The gap endpoints are the endpoints in W_{R6} and in W_{R8} which are closest to p_7. The gap vertex is the midpoint between these two endpoints.

The remaining descriptor data that needs to be calculated is the time at which the vertex was traversed and the angle and width of each leg of the traversed junction. The time at which the vertex was traversed is the time associated with the "DURING" pose, which in our example is p_7. This is the pose at which the robot was actually traversing the gap.

The angle and width of the traversed legs are derived from the wall segment coordinates of the wall segments adjacent to the gap. In our example, the two legs are the leg that spans poses p_4 to p_6 and the leg that spans poses p_8 to p_{10}. For the leg that spans poses p_4 to p_6, these wall data variables store the relevant information: W_{R4}, W_{R5}, W_{R6}, W_{L4}, W_{L5}, and W_{L6}. Each of these variables contains the endpoints and the normal vector for the particular wall segment. Many different geometric approaches could be used to derive the angle and width of this leg. One possible approach is to find a best fit line segment to the right-wall segments (W_{R4}, W_{R5}, W_{R6}) and a best fit line segment to the left wall segments (W_{L4}, W_{L5}, W_{L6}). Then, the normal angle of the leg would be the mean of the normal's of the two best fit line segments, and the width of the leg would be the mean distance between the two best fit line segments.

At this point, we have derived all the descriptor parameters that we need for the T-junction descriptor. These parameters are summarized in table 4.

Table 4
T-junction descriptor parameters

The fact that the junction is a T-junction
Vertex coordinates
Coordinates of the start and end points of the gap
Angle of each traversed leg of the junction
Width of each traversed leg of the junction
Time when gap was traversed

The process for deriving the parameter values for a T-junction is summarized in figure 15.

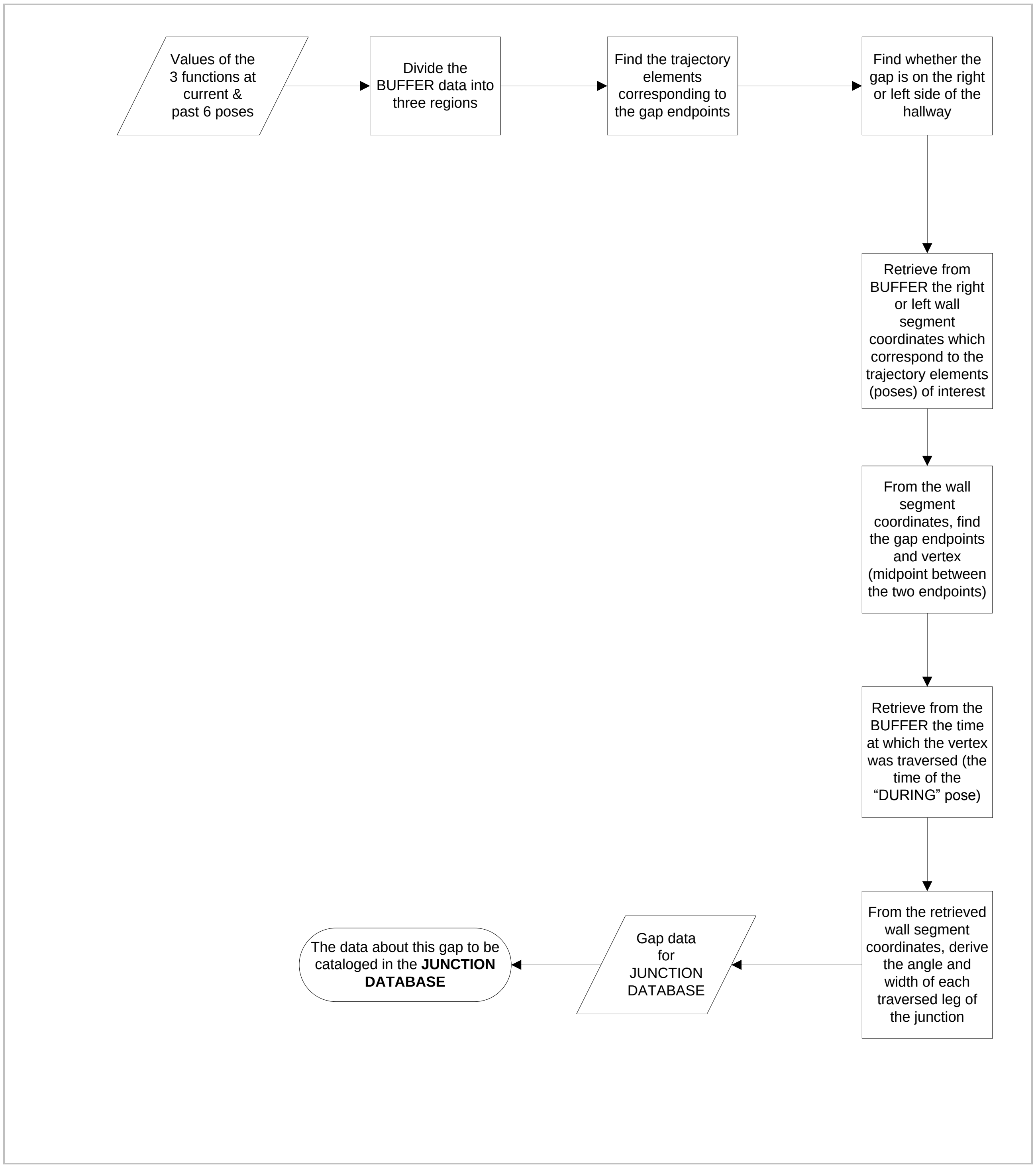

Values of the 3 functions at current & past 6 poses
Divide the BUFFER data into three regions
Find the trajectory elements corresponding to the gap endpoints
Find whether the gap is on the right or left side of the hallway
Retrieve from BUFFER the right or left wall segment coordinates which correspond to the trajectory elements (poses) of interest
From the wall segment coordinates, find the gap endpoints and vertex (midpoint between the two endpoints)
Retrieve from the BUFFER the time at which the vertex was traversed (the time of the "DURING" pose)
From the retrieved wall segment coordinates, derive the angle and width of each traversed leg of the junction
Gap data for JUNCTION DATABASE
The data about this gap to be cataloged in the JUNCTION DATABASE

Figure 15
Procedure for calculating T-junction descriptor parameters

L-Junction Descriptor Derivation

The algorithm reaches this point if and only if neither wall on the sides of the current pose has a gap in it. In this case, there are two possibilities – either the hallway is straight at this point, or it has an L-junction, i.e., a simple turn with no side paths. Therefore, the next task is to determine which possibility is the case. This is determined by calculating the wall angles for the right and left walls.

The overall procedure for doing this is as follows. Each element of the buffer is examined to search for the presence of corners, i.e., nonstraight sections of the walls (recall that the buffer contains wall data collected at the current pose and at the past ten poses). The algorithm searches for corners by examining the wall data from each pair of adjacent poses in the buffer to look for deviations from straightness. For this discussion, let $\hat{n}_{Li}$ be the unit normal vector for wall segment $W_{Li,}$ and let $\hat{n}_{Ri}$ be the unit normal vector for wall segment W_{Ri}. Figure 16 uses this notation to illustrate these definitions of wall normal vectors.

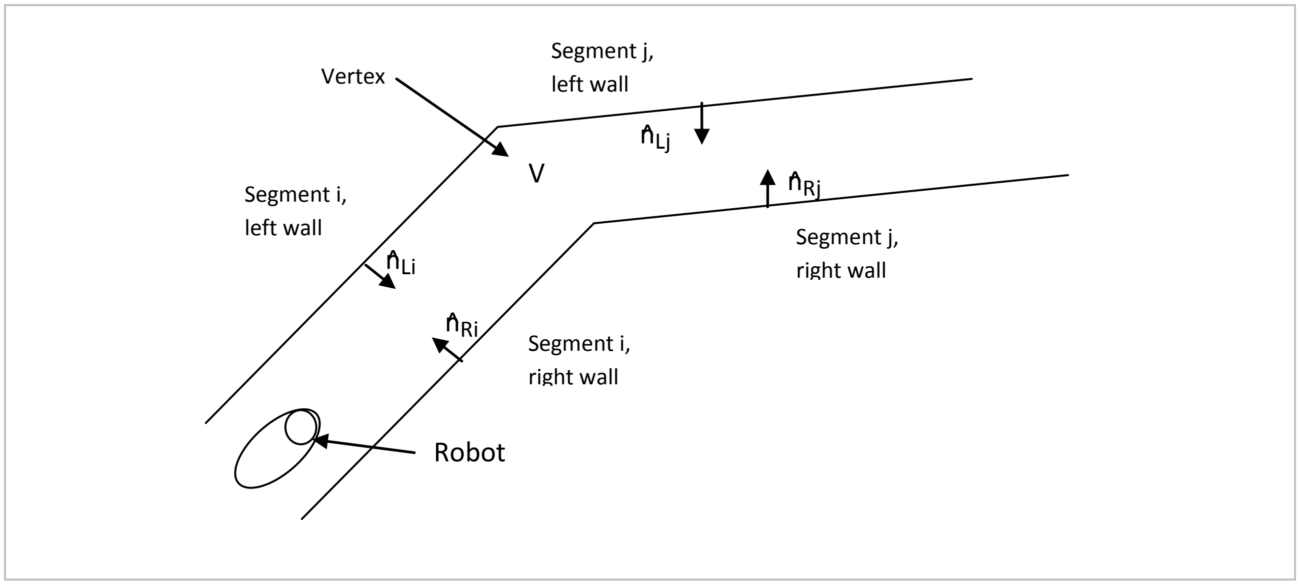


Figure 16
Wall normal vectors for path straightness calculation

The algorithm calculates the difference vector between the normal's for each pair of consecutive right wall segments, and same for the left wall segments. The squared magnitude of the right and left difference vectors is

$$\Delta_{\text{ij, right}} = |\hat{n}_{Ri} - \hat{n}_{Rj}|^2 \tag{12}$$

$$\Delta_{\text{ij, left}} = |\hat{n}_{Li} - \hat{n}_{Lj}|^2 \tag{13}$$

These difference vectors are calculated for each pair of adjacent wall segments in the buffer. The algorithm determines whether, for any pair of adjacent wall segments, both $\Delta_{\text{ij, right}}$ and $\Delta_{\text{ij, left}}$ are larger than the user-specified threshold value *cornerAngleThreshold*.

If no such adjacent pair of wall segments has difference vector magnitudes this large, then we conclude that the buffer contains no corners, and the robot continues traveling with no further calculation being done at the current pose. Otherwise, the next task is to derive the descriptor parameter values for each found corner that has not already been cataloged in the Junction

Database. For each found corner, the following parameters are calculated: location of the vertex, the time it was traversed, the width of each leg of the corner, and the angle between the two legs of the corner.

The location of the vertex of the corner is illustrated by the "V" in figure 16. In particular, the vertex is the midpoint between (1) the intersection point between **right** wall segments *i* and *j*, and (2) intersection point between **left** wall segments *i* and *j*. Just as the location of the vertex is determined through interpolation, so is the time of vertex traversal. That time is defined as the midpoint between the times of pose *i* and of pose *j*.

The next task is to find the widths of each leg of the corner. The endpoints of wall segments *i* and *j* can be retrieved from the elements W_{Ri} and W_{Li} for leg *i*, and from W_{Rj} and W_{Lj} for leg *j*. The width of each leg can be found from this data in a variety of ways. One way to do it is to find the average distance between the two walls associated with pose *i*, and the same with pose *j*.

Finally, the angle between the corner's two legs, as well as the orientation of each leg, has to be determined. This angle can be determined from the normal vectors for the wall segments. The corner angle is the average between two angles: (1) the angle between the normal vectors of the left-side walls, i.e., angle ($\hat{n}_{Li}$, $\hat{n}_{Lj}$), and (2) the angle between the normal vectors of the right side walls, i.e., angle ($\hat{n}_{Ri}$, $\hat{n}_{Rj}$). Viewing figure 16 can make these expressions easier to understand. The orientation of leg *i* is the average of the normal vectors of the right and left walls at pose *i*, i.e., $\frac{1}{2}(\hat{n}_{Li} + \hat{n}_{Ri})$. Similarly, the orientation of leg j is $\frac{1}{2}(\hat{n}_{Lj} + \hat{n}_{Rj})$.

At this point, all the descriptor parameters for a particular L-junction corner have been calculated, and the descriptor is cataloged in the Junction Database. Table 5 summarizes the parameters that are cataloged for an L-junction.

Table 5
L-junction descriptor parameters

The fact that the junction is an L-junction
Vertex coordinates
Vertex angle
Leg orientations
Leg widths
Time when corner was traversed

Now that the descriptor data for a particular L-junction has been calculated and cataloged, the algorithm can go on to calculate these same parameters for the next corner until all corners observed at the current pose have been cataloged. The process for deriving the parameter values for an L-junction is summarized in figure17.

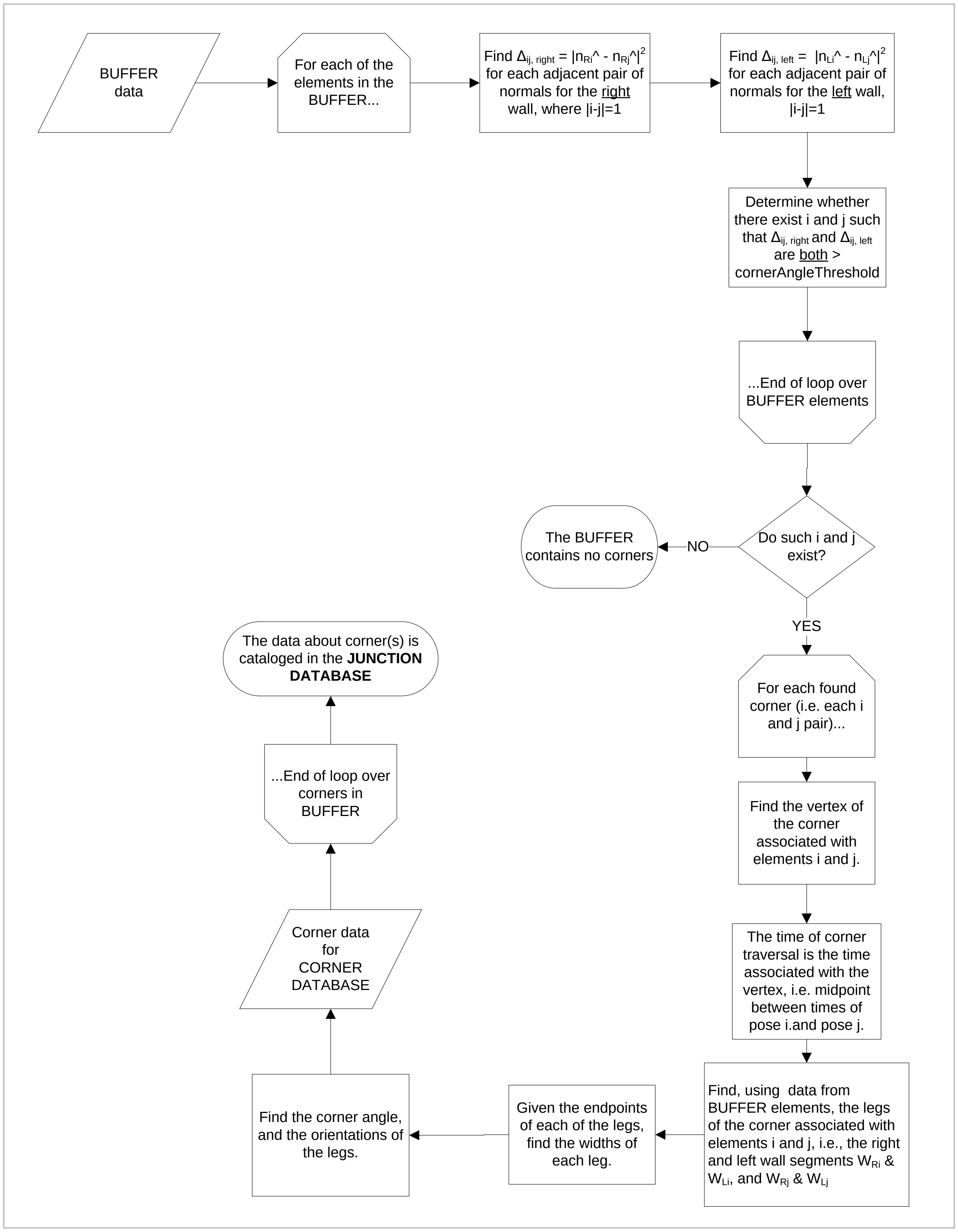

BUFFER data
For each of the elements in the BUFFER...
Find $\Delta_{ij,\,right} = |n_{Ri}\hat{} - n_{Rj}\hat{}|^2$ for each adjacent pair of normals for the right wall, where |i-j|=1
Find $\Delta_{ij,\,left} = |n_{Li}\hat{} - n_{Lj}\hat{}|^2$ for each adjacent pair of normals for the left wall, |i-j|=1
Determine whether there exist i and j such that $\Delta_{ij,\,right}$ and $\Delta_{ij,\,left}$ are both > cornerAngleThreshold
...End of loop over BUFFER elements
Do such i and j exist?
NO
The BUFFER contains no corners
YES
For each found corner (i.e. each i and j pair)...
Find the vertex of the corner associated with elements i and j.
The time of corner traversal is the time associated with the vertex, i.e. midpoint between times of pose i.and pose j.
Find, using data from BUFFER elements, the legs of the corner associated with elements i and j, i.e., the right and left wall segments W_{Ri} & W_{Li}, and W_{Rj} & W_{Lj}
Given the endpoints of each of the legs, find the widths of each leg.
Find the corner angle, and the orientations of the legs.
Corner data for CORNER DATABASE
...End of loop over corners in BUFFER
The data about corner(s) is cataloged in the JUNCTION DATABASE

Figure 17
Procedure for calculating L-junction descriptor parameters

Descriptor Comparison Process for LT-Junction Based Algorithm

At this point in the algorithm, we have just cataloged in the Junction Database whichever type of junction that has just been found – either a T-junction or an L-junction. Table 6 summarizes the parameter values that are cataloged for each type of junction.

Table 6
Summary of descriptor parameters for L- and T-junctions

L-junction	T-junction
Vertex coordinates	Vertex coordinates
Corner angle	Coordinates of the start and end points of the gap
Leg orientations	Angle of each traversed leg of the junction
Leg widths	Width of each traversed leg of the junction
Time when corner was traversed	Time when gap was traversed

Before comparing the newly cataloged junction with previously cataloged junctions in the Junction Database, it is useful to narrow down the set of junctions in the database that will be used in the comparison. Therefore, as done previously in the trajectory based algorithm, a search region is defined which is a subset of previously calculated descriptors to be used for descriptor comparisons. This subset would consist of the set of cataloged descriptors that are most likely to match the current descriptor. In the following discussion, the following terminology is used: The “target descriptor” is denoted as C^T, and the descriptor in the Junction Database with which the target descriptor is being compared is denoted as the “candidate descriptor” C^C.

The first criterion for narrowing down the search set is that only candidate descriptors of the same type as the target descriptor are searched for. That is, if the target descriptor is for an L-junction, then only candidate junctions that are also L-junctions are considered, and similarly for T-junctions. After this initial screening process, the criteria listed in equations 3, 4, 5, and 6 are used to select which descriptors in the Junction Database are most likely to match the current descriptor and will therefore be compared in detail with the current descriptor.

Now that the four criteria for the search region have been applied, the Junction Database is examined to determine which, if any, candidate descriptors in it satisfy all of the search region criteria. If no candidates satisfy all four criteria, then no further calculations are done at this step in the trajectory, and the robot travels further.

However, if some candidates that do satisfy all four criteria are found, then these candidate descriptors are examined in order to determine how closely they match the target descriptor. Figure 18 shows the procedure for scoring the match between the target and candidate corners.

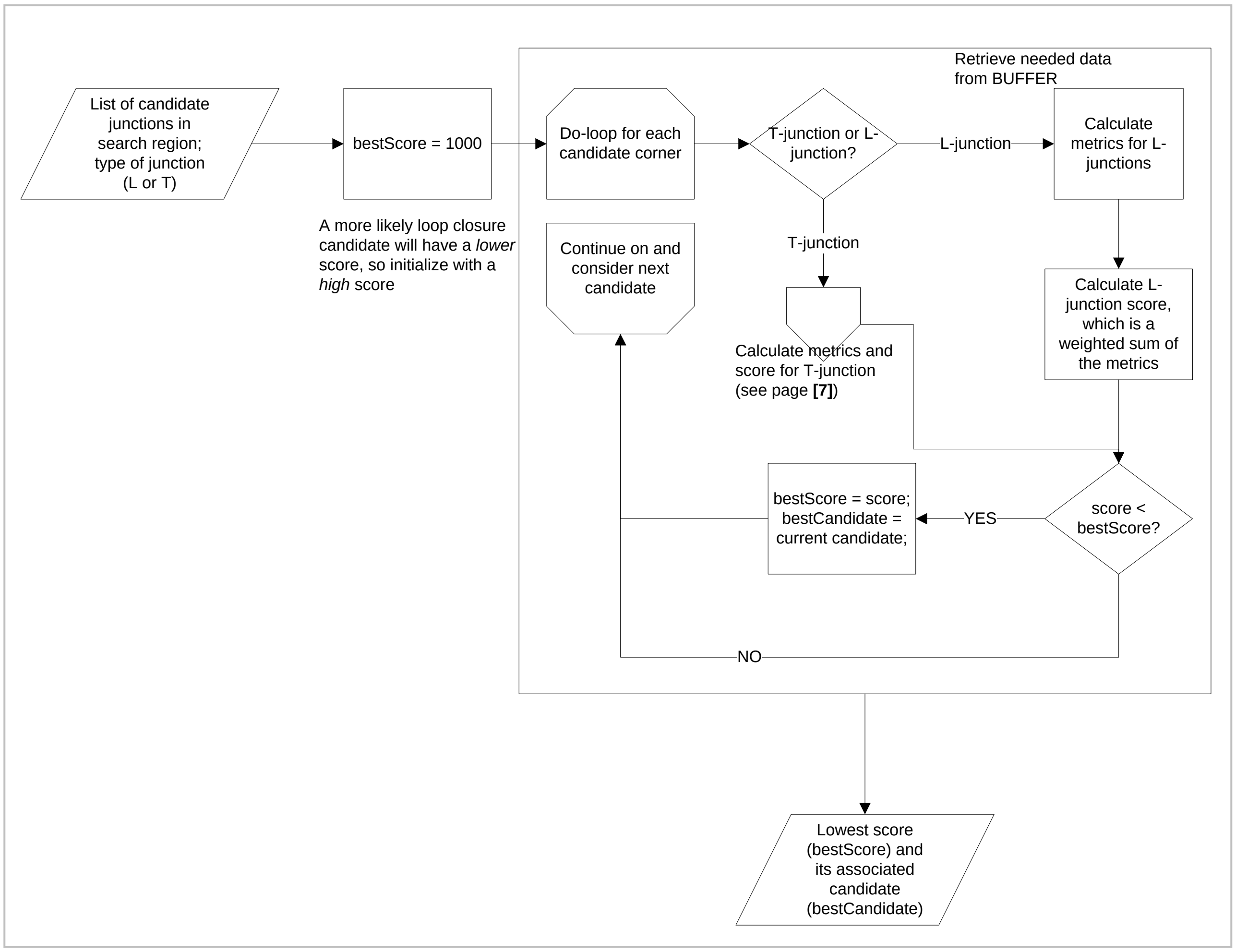

List of candidate junctions in search region; type of junction (L or T)
bestScore = 1000
A more likely loop closure candidate will have a *lower* score, so initialize with a *high* score
Retrieve needed data from BUFFER
Do-loop for each candidate corner
T-junction or L-junction?
L-junction
Calculate metrics for L-junctions
T-junction
Continue on and consider next candidate
Calculate metrics and score for T-junction (see page **[7]**)
Calculate L-junction score, which is a weighted sum of the metrics
bestScore = score; bestCandidate = current candidate;
YES
score < bestScore?
NO
Lowest score (bestScore) and its associated candidate (bestCandidate)

Figure 18
Scoring method for evaluation of target and candidate match for LT-junction based algorithm

The input to this process is the list of candidate junctions in the search region which are of the same type (T or L junction) as the target junction. Each candidate junction is scored according to the following process, first, the metrics and score for the match between the two junctions are found. This process is rather complex, and is described in detail in the next two sections – one for L-junctions and one for T-junctions. Once the score is calculated for the currently considered candidate, then it is compared with the best score of all the previously considered candidates at this step in the trajectory. After all candidates have been considered, the output data for the candidate with the lowest score is calculated. The output data of this method is derived from the descriptor data of the selected candidate junction and of the target junction. This output data is the vertices of the two junctions, and later in the algorithm it will be passed to a trajectory adjustment program.

The next two sections describe the details of the two sets of similarity tests to be used for this purpose – one set for L-junctions and one for T-junctions.

L-Junction Similarity Tests and Scoring

For L-junctions, four similarity tests are performed on the candidates to determine their similarity to the target descriptor: (1) proximity of vertices, (2) similarity of corner angles, (3) similarity of corner orientations, and (4) similarity of junction leg widths. For each of these tests, a metric is calculated, and then a matching score is calculated as a weighted sum of the four metric values. The metrics and matching score are defined in such a way that the lower the value, the better the match. The four metrics are defined

$$\text{proximityMetric} = \%\text{diff}\ (\text{coordinates of target's vertex, coordinates of candidate's vertex}) \tag{14}$$

$$\text{cornerAngleMetric} = \%\text{diff}\ (\text{target corner angle, candidate corner angle}) \tag{15}$$

In these equations, the %diff function is defined

$$\%\text{diff}\ (a, b) = 2\ |a - b|\ /\ (a + b) \tag{16}$$

$$\text{cornerOrientationMetric} = \%\text{diff}(\text{angle of centerline of target corner, angle of centerline of candidate corner}) \tag{17}$$

Here, the angle of the centerline of a corner is the average of the angles of the two legs of the corner. The fourth metric for comparisons of L-junctions is

$$\begin{aligned} \text{cornerLegWidthMetric} &= \\ \tfrac{1}{2} * &(\%\text{diff}\ (\text{width of lower angle leg of target corner, width of lower angle leg of candidate corner}) \\ &+ \%\text{diff}\ (\text{width of higher angle leg of target corner, width of higher angle leg of candidate corner})) \end{aligned} \tag{18}$$

The first three of these metrics are the same as the similarity metrics for the trajectory based algorithm. The fourth metric deals with the legs of the corner, which is not relevant to the trajectory based algorithm. This metric compares the two corners leg by leg and takes the average of the percent difference of the two leg widths.

The similarity score for the target and candidate corners is a weighted sum of the four different similarity metrics. A suggested weighting is

$$\begin{aligned} &\text{score} = \\ &0.4 * \text{proximityMetric} + \\ &0.3 * \text{cornerAngleMetric} + \\ &0.15 * \text{cornerOrientationMetric} + \\ &0.15 * \text{cornerLegWidthMetric} \end{aligned} \tag{19}$$

T-Junction Similarity Tests and Scoring

T-junctions are much more complex than L-junctions. An L-junction has only two legs, and both of them are traversed by the robot. However, a T-junction has three legs, only two of which are traversed by the robot. The third leg of a T-junction is not traversed, so all that is known about it is that there is a gap in the wall at the location of that leg. These concepts can be better visualized by viewing figures 11 and 12.

The comparison process between two T-junctions is rather complex. The inputs to the process are a target T-junction and a candidate T-junction. First, the proximity metric is calculated for the vertices of the two junctions.

For the second metric, the two traversed legs of each junction are compared. If we call the two traversed legs in each junction leg *i* and leg *j*, then the following four leg pairs are compared to obtain preliminary scores

target leg *i* and candidate leg *i*

target leg *i* and candidate leg *j*

target leg *j* and candidate leg *i*

target leg *j* and candidate leg *j*

Each of these four leg pairs is compared in two ways – the leg angle and the leg width. In particular, for each leg pair, the percent difference for each of the two comparisons is calculated and added together. Both legs of the leg pair with the lowest preliminary score (recall that a low score indicates a good match) are tagged. That is, they are tentatively assumed to match each other. The second metric records this lowest preliminary score.

The third and fourth metrics determine how well the untagged leg and gap of the target and candidate junctions compare. The third metric records the comparison between the target gap and the untagged candidate leg, and the fourth metric records the comparison between the candidate gap and the untagged target leg.

These four metrics are listed in equation form

$$\begin{aligned}&\text{proximityMetric} =\\ &\%\text{diff (coordinates of target junction's vertex, coordinates of candidate}\\ &\text{junction's vertex)}\end{aligned} \tag{20}$$

$$\begin{aligned}&\text{sharedLegMatchMetric} =\\ &\%\text{diff (angle of target traversed leg, angle of candidate traversed leg)} +\\ &\%\text{diff (hallway width of target traversed leg, hallway width of candidate}\\ &\text{traversed leg)}\end{aligned} \tag{21}$$

$$\begin{aligned}&\text{targetGapCandidateLegMatchMetric} =\\ &\%\text{diff (target gap width, candidate leg width)}\end{aligned} \tag{22}$$

$$\begin{aligned}&\text{candidateGapTargetLegMatchMetric} =\\ &\%\text{diff (candidate gap width, target leg width)}\end{aligned} \tag{23}$$

The similarity score for the target and candidate corners is a weighted sum of the four different similarity metrics. A suggested weighting is

$$\begin{aligned} &\text{score} = \\ &0.4 * \text{proximityMetric} + \\ &0.3 * \text{sharedLegMatchMetric} + \\ &0.15 * \text{targetGapCandidateLegMatchMetric} + \\ &0.15 * \text{candidateGapTargetLegMatchMetric} \end{aligned} \tag{24}$$

The process for calculating the metrics used to derive this matching score for T-junctions is shown in figure 19.

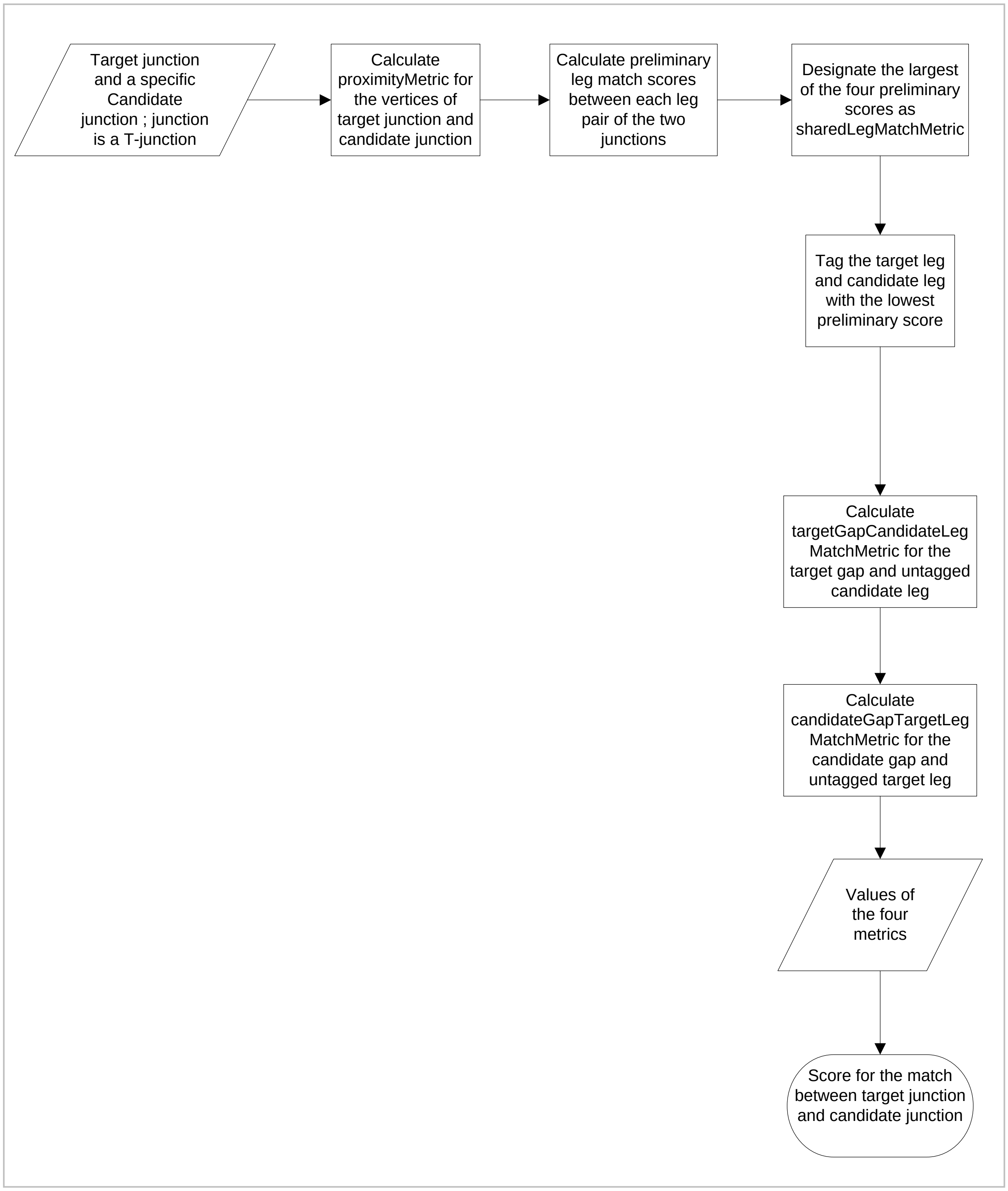

Target junction and a specific Candidate junction ; junction is a T-junction
Calculate proximityMetric for the vertices of target junction and candidate junction
Calculate preliminary leg match scores between each leg pair of the two junctions
Designate the largest of the four preliminary scores as sharedLegMatchMetric
Tag the target leg and candidate leg with the lowest preliminary score
Calculate targetGapCandidateLeg MatchMetric for the target gap and untagged candidate leg
Calculate candidateGapTargetLeg MatchMetric for the candidate gap and untagged target leg
Values of the four metrics
Score for the match between target junction and candidate junction

Figure 19
Procedure for calculating matching score for T-junctions

Derivation of Output Data for LT-Junction Based Algorithm

The outputs of the scoring method for evaluation of target and candidate matches, shown in figure 18, are the candidate that best matches the target junction and the matching score. At this point in the algorithm, the next step is to compare this matching score with a user-specified threshold value, *smax*. If the score is greater than *smax*, i.e., worse than the threshold value, then no further calculations are performed at this step of the trajectory, and the robot travels further. Otherwise, the output data of the entire algorithm is calculated.

For the LT-junction based algorithm, the output data is simply the (x, y) coordinates of the two vertices – the vertex of the target junction and the vertex of the selected candidate junction, as well as the matching score. This is the output regardless of whether the junction is an L-junction or a T-junction. Orientation data from the descriptors is not part of the output since it is up to the trajectory adjustment program (to which data from this algorithm is passed) to figure out the optimum orientation of the target pose and candidate pose in the adjusted map. The complete set of output data for the LT-junction based algorithm is shown in table 7.

Table 7
Output data for LT-junction based algorithm

Target descriptor
Candidate descriptor
Matching score, *score*

Overall Summary of LT-Junction Based Algorithm

Table 8 summarizes the user-selectable parameters for the LT-junction based algorithm.

Table 8
List of selectable parameters for LT-junction based algorithm

Parameter description	**Variable name**	**Assumed value, if any**
Robot's step distance	Δp	
Number of elements in buffer		11
Threshold for nearest wall distance function in gap detection for T-junctions	*deltaWidth*	
Threshold for difference in wall segment angle difference in L-junction detection	*cornerAngleThreshold*	
Minimum time for search region	t_{min}	
Maximum time for search region	t_{max}	
Minimum distance for search region	d_{min}	
Maximum distance for search region	d_{max}	
Threshold score for place recognition candidates	s_{max}	

Figure 20 shows a high level overview of the entire LT-junction based algorithm. An advantage of this algorithm is that it makes use of wall data that is available in walled environments to help make a place recognition decision. This is in contrast to the trajectory based algorithm, which does not make any use of wall data. A limitation of the LT-junction based algorithm is that it deals only with L-junctions, T-junctions, and by extension, +-junctions. However, it cannot reliably deal with more complex junctions, i.e., those with five or more legs intersecting at a single vertex.

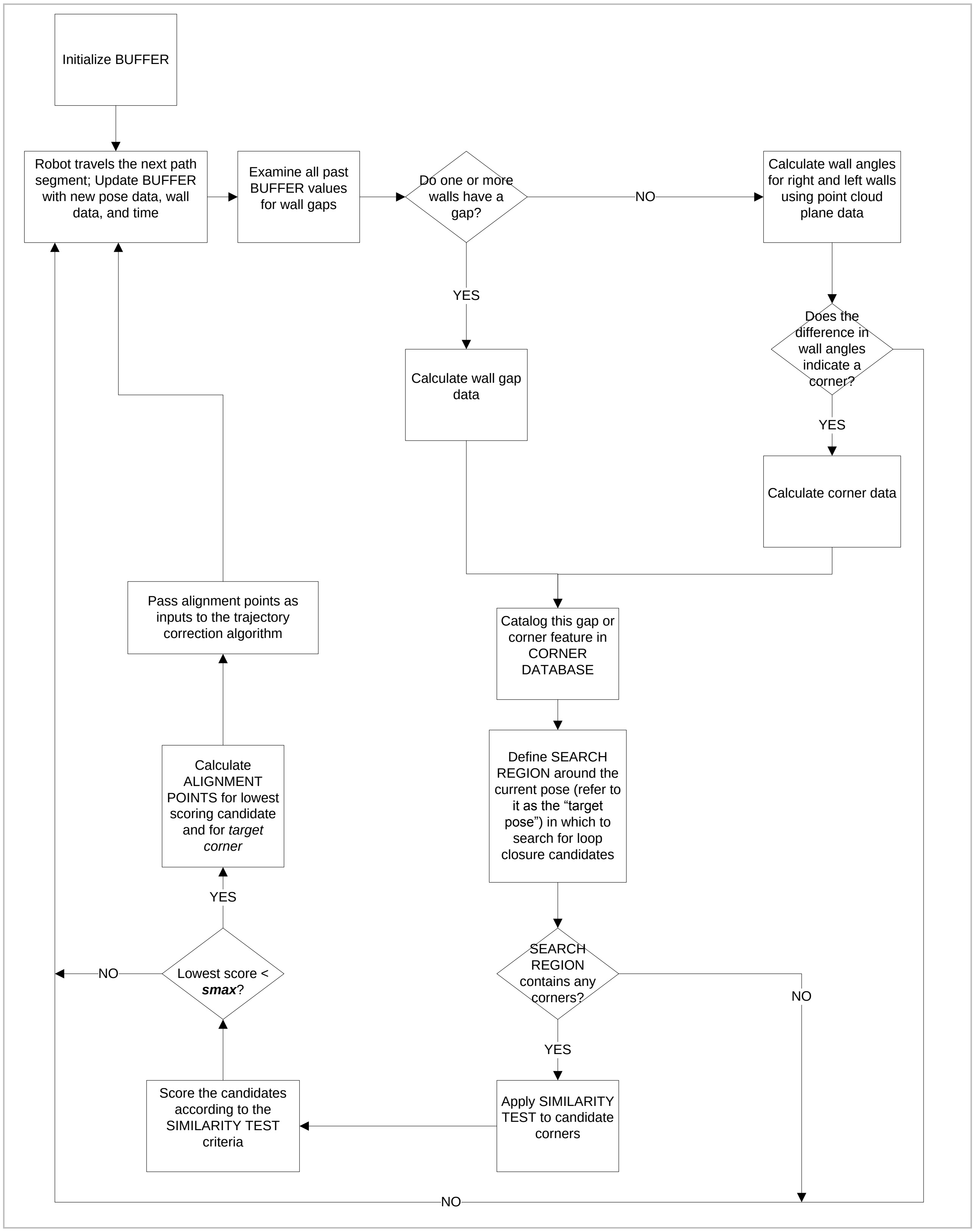

Initialize BUFFER
Robot travels the next path segment; Update BUFFER with new pose data, wall data, and time
Examine all past BUFFER values for wall gaps
Do one or more walls have a gap?
NO
Calculate wall angles for right and left walls using point cloud plane data
YES
Does the difference in wall angles indicate a corner?
YES
Calculate wall gap data
Calculate corner data
Catalog this gap or corner feature in CORNER DATABASE
Pass alignment points as inputs to the trajectory correction algorithm
Define SEARCH REGION around the current pose (refer to it as the "target pose") in which to search for loop closure candidates
Calculate ALIGNMENT POINTS for lowest scoring candidate and for *target corner*
YES
NO
Lowest score < ***smax***?
SEARCH REGION contains any corners?
NO
YES
Score the candidates according to the SIMILARITY TEST criteria
Apply SIMILARITY TEST to candidate corners
NO

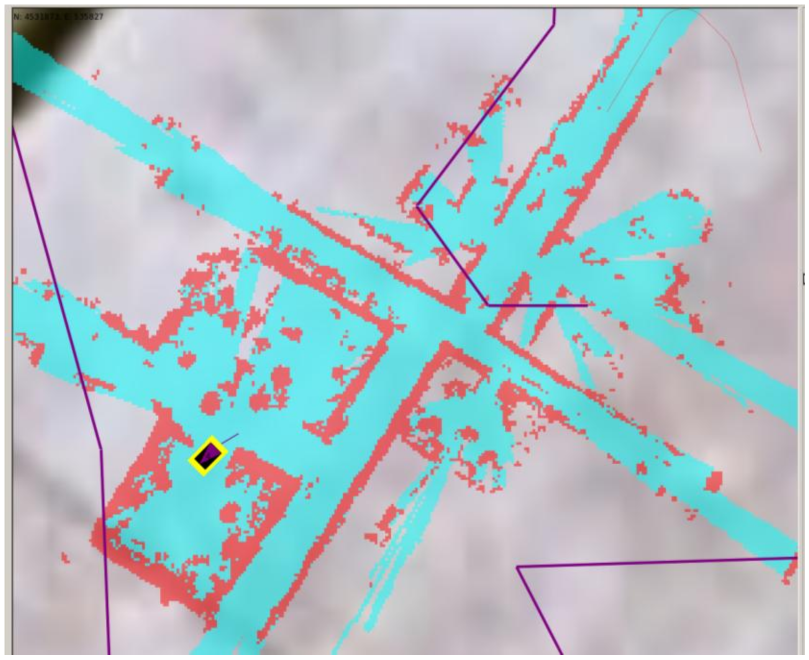
Figure 20
High level flow chart for LT-junction based algorithm

General Junction Based Algorithm

This section describes, for the general junction based algorithm, the input data and its preprocessing, the feature descriptor and its derivation from the preprocessed input data, the descriptor comparison process, and the derivation of output data.

The general junction based algorithm is useful for place recognition and loop closure detection in situations in which a robot is driving in a network of featureless passageways, with nominally flat, parallel walls, that constrains its motion to either nominally straight line driving or turning from one path segment onto another. It uses hallway junction characteristics to label and compare visited places and to search for recognized places.

This algorithm detects junctions with, in principle, any number of hallway paths connected to them. Unlike the trajectory based algorithm and the LT-junction based algorithm, this algorithm does not maintain a sliding buffer to enable detection of junction features. Instead, it takes as an input a visual 2D image of a LIDAR scan of the region that the robot has traversed until the present time. An example is shown in figure 21.



Note: Figure supplied by Joshua Wainer of Robotic Research, LLC

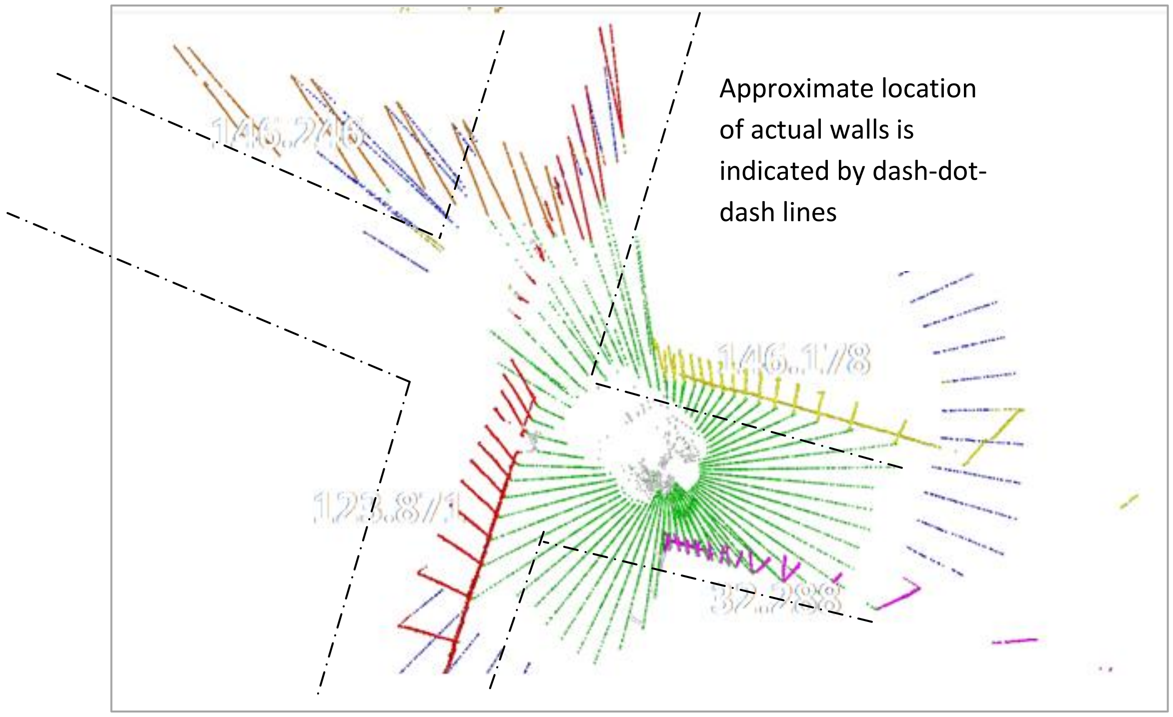
Figure 21
Typical input image for general junction based algorithm

The algorithm uses various image processing methods to extract junctions from the input image. It then catalogs each junction it detects into a database, and compares newly detected junctions to the junctions in the database in order to determine whether a place recognition event has occurred.

Input Data and Preprocessing for General Junction Based Algorithm

As with the trajectory based algorithm and the LT-junction based algorithm, this algorithm uses 2D data as its input. In some ways, the general junction based algorithm is similar to the LT-junction based algorithm; both algorithms use LIDAR sensor data as their source of information about environmental features. Both algorithms use wall data as descriptors of the environment. However, the two algorithms use different representations of wall data.

The LT-junction based algorithm represents walls in terms of distance from the robot to nearby walls and lengths of nearby wall segments as measured by a LIDAR sensor. Open source point cloud processing software (ref. 14) was considered for use in implementing this algorithm, specifically for identifying wall segments and determining the distance to them. There are some significant disadvantages to obtaining wall descriptor data directly from point cloud data. For long-exposure (more than 1 sec or so) point cloud images, the wall edges appear fuzzy due to random fluctuations in the LIDAR data. It is difficult to identify wall edges in such images using point cloud data processing techniques. On the other hand, short exposure point cloud images exhibit clearly defined walls but data on these walls is quite sparse, and that makes it difficult to define the endpoints, and even the orientation, of wall segments. Figure 22 shows a typical example of a short exposure point cloud image of a hallway junction.


Approximate location of actual walls is indicated by dash-dot-dash lines
146.216
146.178
123.871
32.288

Note: Figure supplied by Joshua Wainer of Robotic Research, LLC

Figure 22
Short exposure image of hallway junction taken by LIDAR

In figure 22, a random sample consensus (RANSAC) approach from the Point Cloud Library was used to extract the walls. In this image, the robot is in the center of the junction facing toward the lower right corner of the image. To facilitate a qualitative evaluation of the image, the approximate location of the actual walls is denoted by dash-dot-dash lines.

Since both long and short exposure images are difficult to analyze using Point Cloud processing approaches, an alternative approach was considered. In this alternative approach, a visual (e.g., jpg format) image is formed from a long exposure Point Cloud image. Then, open source visual processing approaches (ref. 15) are used to extract hallway junction data from the visual image. This approach is advantageous because a mature and robust set of image processing procedures is readily available and could effectively extract junction information from the visual images. In the remainder of this section, the image processing methods used to derive corner descriptors are presented.

A Gaussian blur (ref. 16) is used as an initial processing step in order to reduce the presence of artifacts in preparation for edge detection procedures. The scale size of the blur would be selected by examining a few typical long exposure Point Cloud images of the area to be mapped (or a different area with similar hallway dimensions) and noting the spatial scales of the hallway widths and of the noise artifacts in the hallway edges.

After the Gaussian blur is performed, the next step is to look for corners in the image. This is done using a Harris corner detector (ref. 17), e.g., by using the function *cornerHarris* in OpenCV. This function detects points with high gradients in both the *x* and *y* directions. The function returns the (x, y) coordinates of detected corners. It is possible that this function would return some spurious corners in addition to the genuine corners. Therefore, it is worthwhile to implement a step to filter out spurious corners, as a possible example (ref. 18) by using the smallest univalue segment assimilating nucleus (SUSAN) corner detector after the harris corner detector. However, even if some spurious corners pass through such a filtering stage, they will quite possibly be filtered out in future steps in the algorithm.

At this point the algorithm determines whether any genuine corners have been detected. If not, then no further calculation is done at this step of the trajectory, and the robot continues driving. If any genuine corners have been detected, then the following steps are performed for each detected corner in order to derive parameter values for the corner descriptor for that corner

1. Define a region around the corner in which to search for line segments connected to that corner. These line segments will be included in the descriptor for this corner. This region should be small compared with the hallway width in order to avoid including segments from other parts of the hallway in the descriptor.

2. Within that region, extract edges connected with the corner. A canny edge detector (ref. 19) from OpenCV can be used for this purpose. This function would return only an image consisting of detected line segments. We still need coordinates of these line segments.

3. Represent edges as line segments and derive their endpoints. A probabilistic Hough transform (ref. 20) from OpenCV can be used for this purpose and gives as its output the (x, y) coordinates of the endpoints of each detected line segment.

4. If any edges connected with the corner are found, record them for immanent use in populating the descriptor for this corner.

Figure 23 illustrates the definitions for the various types of hallway feature that may be detected like corners, vertices, and wall segments.

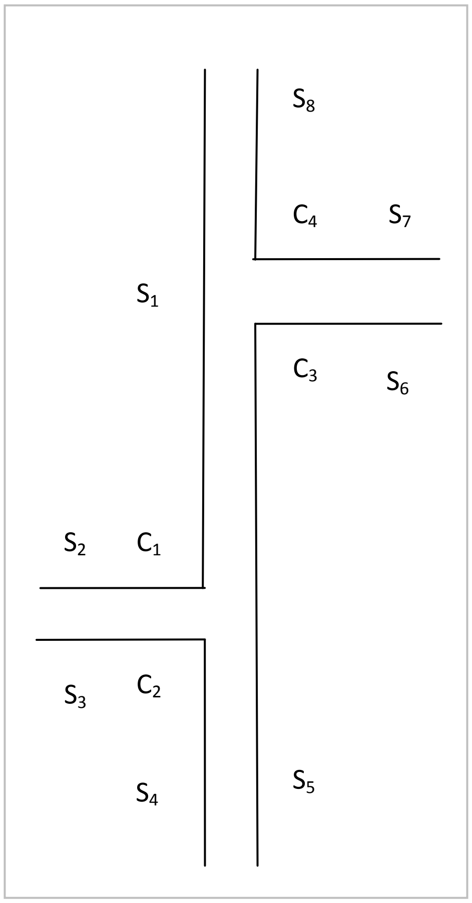


Figure 23
Terminology for hallway features

A corner is treated as an object comprised of a vertex and two segments. A vertex is an (x, y) point. A side is a data structure comprised of two (x, y) endpoints. In the hallway configuration example shown in figure 23, there are four corners: corner C_1, comprised of vertex V_1 and sides S_1 and S_2; corner C_2 comprised vertex V_2 and sides S_3 and S_4; corner C_3, comprised vertex V_3 and sides S_5 and S_6; and corner C_4, comprised vertex V_4 and sides S_7 and S_8. In table form, the set of corners in figure 23 would be depicted as shown in table 9.

Table 9
Summary of corner objects in figure 23

C_1	**C_2**	**C_3**	**C_4**
V_1	V_2	V_3	V_4
S_1	S_3	S_5	S_7
S_2	S_4	S_6	S_8

The steps involved in the image processing stage of the algorithm are summarized in figure 24.

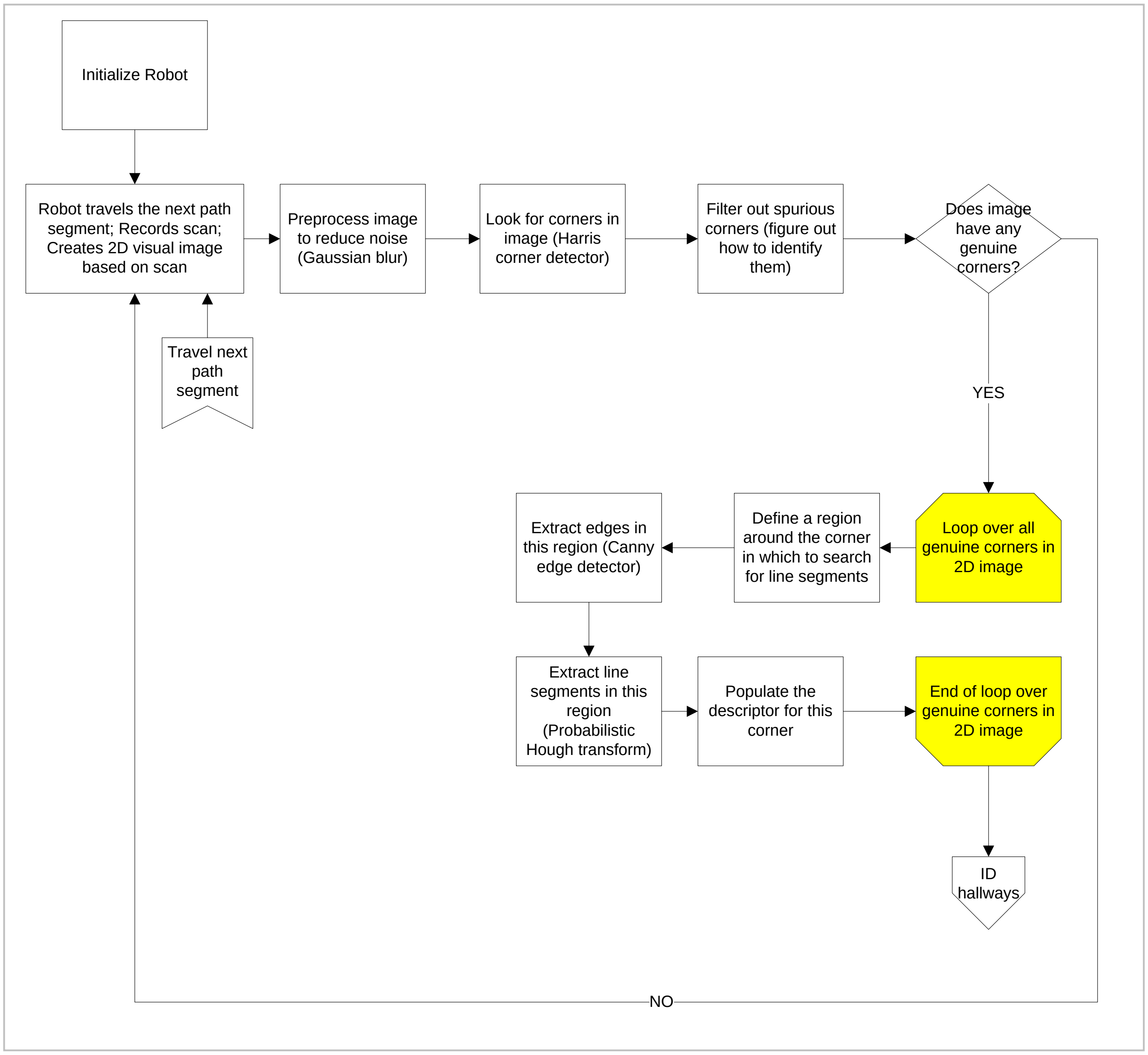

Initialize Robot
Robot travels the next path segment; Records scan; Creates 2D visual image based on scan
Preprocess image to reduce noise (Gaussian blur)
Look for corners in image (Harris corner detector)
Filter out spurious corners (figure out how to identify them)
Does image have any genuine corners?
YES
NO
Travel next path segment
Loop over all genuine corners in 2D image
Define a region around the corner in which to search for line segments
Extract edges in this region (Canny edge detector)
Extract line segments in this region (Probabilistic Hough transform)
Populate the descriptor for this corner
End of loop over genuine corners in 2D image
ID hallways

Figure 24
Summary of image processing stage of general junction based algorithm

Feature Descriptor and its Derivation for General Junction Based Algorithm

At this point in the algorithm, all corners in the current scan have presumably been identified and cataloged as shown in table 9. The next task is to search the identified corners to find hallways.

To accomplish this, the descriptors for each pair of detected corners are compared to determine whether there is a pair of sides, one from each corner, which together form a hallway. Each pair of corners C_i and C_j, is examined to determine whether it complies with all of the following hallway criteria. These criteria indicate the presence of a corner plus a hallway comprised of one side from the corner

- At least one segment from C_i is parallel to at least one segment from C_j, and
- The two parallel segments are at least partially side-by-side (fig. 25), and
- The two parallel segments are close enough to be considered a hallway, according to a user-specified threshold *hallWidthThreshold*, which is the maximum distance between two line segments for them to be considered a hallway

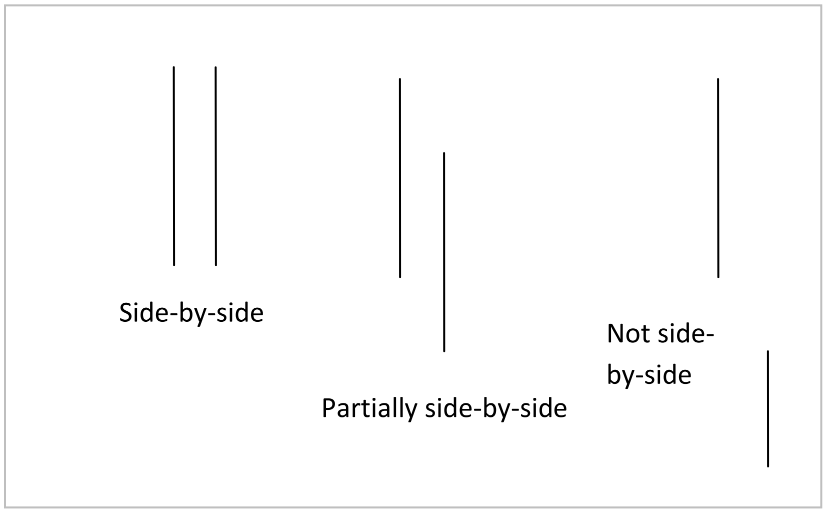


Figure 25
Illustration of side-by-side segments

The result of applying the hallway criteria to the hallway configuration in figure 25 is the following list of hallways

H_{23}: $S_2 - S_3$

H_{15}: $S_1 - S_5$

H_{45}: $S_4 - S_5$

H_{67}: $S_6 - S_7$

H_{81}: $S_8 - S_1$

The purpose of the next section of the algorithm is to determine data for, and catalog, the junction descriptors. Each junction J_i is an object comprised[3] of the junction centroid (JC_i), and the centerline segments $\{L_{ij}\}$ for each of the *j* legs. The junction centroid is an (x, y,) point. Each centerline segment is a data structure comprised of the centerline's two (x, y) endpoints. At this point in the algorithm, the centroid and legs of the junction must be determined. If the scan contains more

[3] The time t_i when the junction was traversed is not included in the descriptor here, although this information would be very useful, because it is not immediately clear how to obtain or preserve it from the image processing stage of the algorithm.

than one junction, then a descriptor is determined and cataloged for each junction. The remainder of this section describes how to derive the data for the junction descriptor.

. The list of hallways that was previously compiled implicitly contains information about junctions. In the example, there is a T-junction that has as its legs H_{81}, H_{67}, and H_{15}, and another T-junction that has as its legs H_{15}, H_{23}, and H_{45}. In general terms, this algorithm must now extract junction information out of the list of hallways. To do this, for simplicity, each hallway segment is first collapsed into the line segment which is its centerline. When the two segments are side-by-side, as defined in figure 25, it is clear how to define the centerline. When the two segments are only partially side-by-side, then the centerline is defined as the centerline between the side-by-side portion of the two segments, as shown in figure 26.

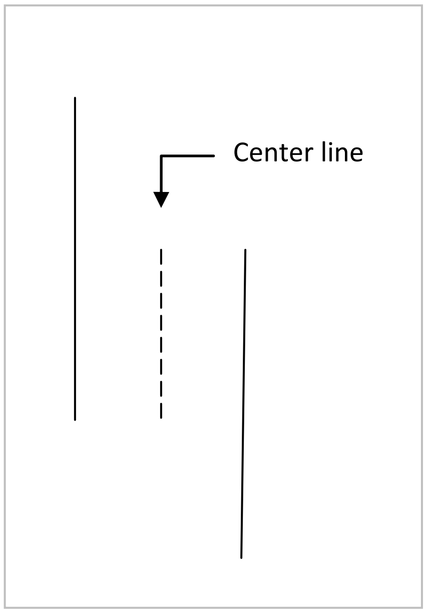


Figure 26
Definition of centerline for partially side-by-side segments

Now that there is a set of centerlines, the two end points of each centerline can be extracted. Once the set of centerline endpoints is established, the algorithm looks for clusters of two or more points that are closer together than the typical hallway width. This closeness indicates the presence of a junction. Figure 27 illustrates the identification of clusters of centerline endpoints in the context of our hallway example and shows how these clusters indicate the presence of junctions.

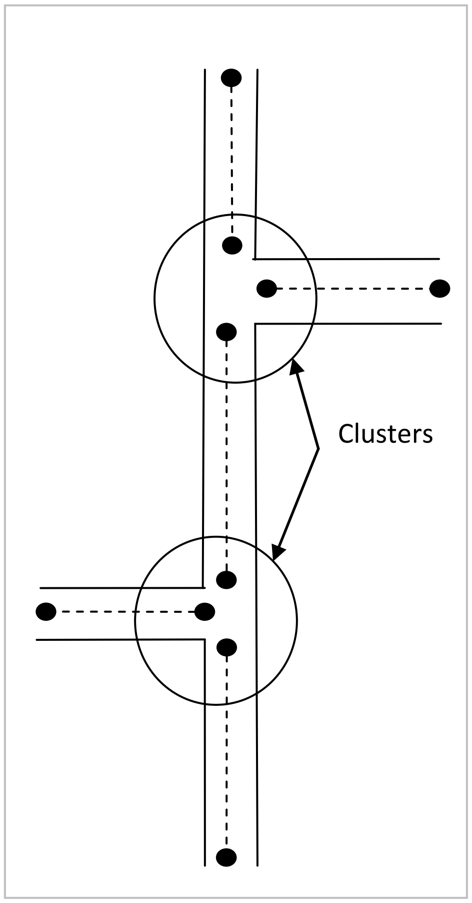


Figure 27
Identification of clusters from centerline endpoints

A user-defined threshold, *clusterThreshold*, specifies the maximum distance between centerline endpoints for the points to be considered part of the same cluster.

Once all the endpoint clusters have been identified, the centroid of each cluster is calculated, and these centroids will be designated as the vertices of junctions. At this point, the algorithm makes note of how many junctions have been identified. This will be important later when comparing the target scan with candidate scans – the comparison process depends on whether there is only one junction in each scan, or whether each scan has multiple junctions.

Now, all the parameters in the junction descriptor can be calculated. To do so, each junction centroid is associated with the legs of the junction, which are the centerline segments connected with each of the centerline endpoints in the cluster from which the centroid was derived. It is possible for a particular centerline to be part of more than one junction. For example, figure 27 shows a vertical line segment that is part of both junctions. The complete junction descriptor(s) is now stored in the Junction Database in preparation for performing junction matching processes. Table 10 shows all the components of the junction descriptor.

Table 10
Junction descriptor parameters for general junction based algorithm

Name of junction, **J_i**
Junction centroid, JC_i
Leg 1 endpoints, L_{i1}
Leg 2 endpoints, L_{i2}
Leg 3 endpoints, L_{i3}
(data on more legs, if any)

The steps involved in the image processing stage of the algorithm are summarized in figure 28.

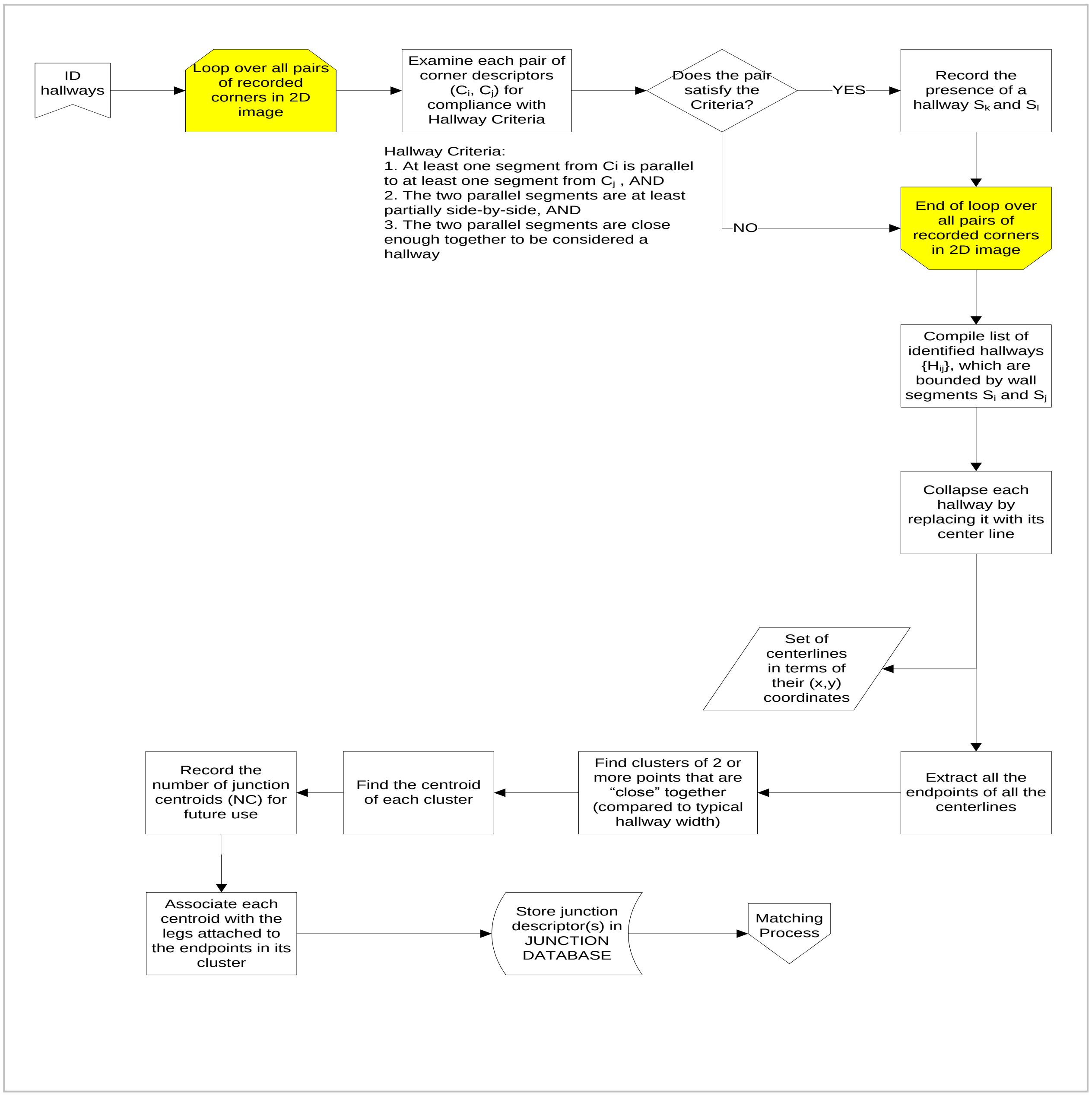

ID hallways
Loop over all pairs of recorded corners in 2D image
Examine each pair of corner descriptors (C_i, C_j) for compliance with Hallway Criteria
Does the pair satisfy the Criteria?
YES
Record the presence of a hallway S_k and S_l
Hallway Criteria:
1. At least one segment from Ci is parallel to at least one segment from C_j , AND
2. The two parallel segments are at least partially side-by-side, AND
3. The two parallel segments are close enough together to be considered a hallway
NO
End of loop over all pairs of recorded corners in 2D image
Compile list of identified hallways {H_{ij}}, which are bounded by wall segments S_i and S_j
Collapse each hallway by replacing it with its center line
Set of centerlines in terms of their (x,y) coordinates
Extract all the endpoints of all the centerlines
Find clusters of 2 or more points that are "close" together (compared to typical hallway width)
Find the centroid of each cluster
Record the number of junction centroids (NC) for future use
Associate each centroid with the legs attached to the endpoints in its cluster
Store junction descriptor(s) in JUNCTION DATABASE
Matching Process

Figure 28
Summary descriptor derivation stage of general junction based algorithm

Descriptor Comparison Process for General Junction Based Algorithm

As in the trajectory based algorithm and the LT-junction based algorithm, only a subset of candidates in the Junction Database is searched for potential matches to the candidate junction(s). This subset consists of candidates whose location is within a specified region of the target junction. In particular, if we define the coordinates of the target junction centroid as ($C^T.x$, $C^T.y$), and the coordinates of the candidate target junction as ($C^C.x$, $C^C.y$), then equations 5 and 6 define the criteria for candidate junctions to fall within the search region. The purpose of the user-specified parameter d_{min} is to filter out from consideration meanderings of the robot about its current position. The purpose of the user-specified parameter d_{max} is to filter out candidate corners that are very distant from the target corner in order to reduce computation time. If the traversal times of the junctions are included in the junction descriptors (see footnote 3), then additional criteria (eqs. 3 and 4) can be used to further narrow down the search region.

The algorithm now determines (as described in further detail later) whether the Junction Database has any junctions within the search region that has just been determined. If the search region has no junctions, then no further calculation is done at this step of the trajectory, and the robot travels further. If the search region does have some junctions in it, then each of these candidate junctions is compared with the target junction to evaluate how well it matches the target junction. In performing these comparisons, the algorithm determine how many junction matches there are, i.e., how many of the junctions in the target scan have well matching scans in the Junction Database.

The similarity test has three possible outcomes: {zero, one, or more than one} target junction(s) has a good match in the Junction Database. If no target junction has a good match in the Junction Database, then no further calculation is done at this step, and the robot travels further. If one target junction has a good match, then the single junction matching procedure (described in detail later) is followed, which yields single junction output data. If more than one target junction has a good match, then the multi junction matching procedure (described in detail later) is followed, which yields single junction output data for each matching pair, as well as a multi junction matching score, which characterizes the overall matching quality of the set of individual matches.

In the multi junction case, it is important to calculate an overall matching score in addition to the individual matching scores, because it is possible that the individual junction pairs are well matched, but the total set of matches is not self-consistent. Figure 29 illustrates such a case.

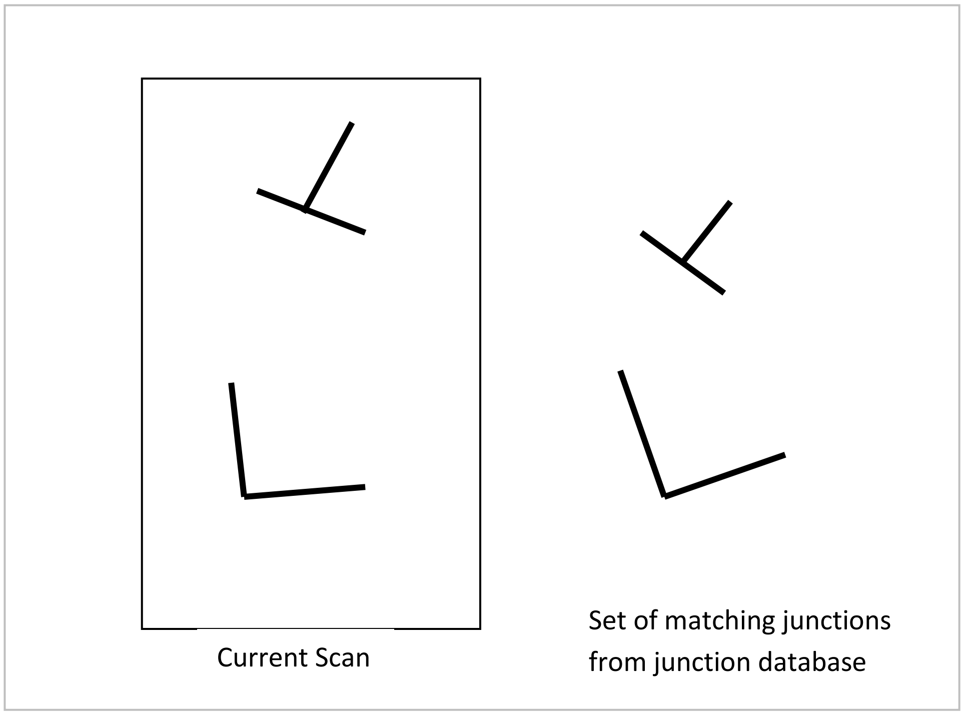


Figure 29
Example of poorly matching multiple scans

In figure 29, the current scan has two junctions, and each has a good match in the Junction Database. Note that the T-junction in the current scan would have to be rotated slightly clockwise to match the corresponding junction from the database; whereas, the L-junction in the current scan would have to be rotated slightly counterclockwise. Thus, each individual match is good, but the two matches are not self-consistent. That is, the T-junction and L-junction in the current scan are close together and have a certain alignment with respect to each other. However, the two junctions in the database do not have the same alignment with respect to each other, although they are not far apart from each other since they are both within the search region. A multi-junction match would show a poor overall matching score between the junctions in the current scan and the set of junctions from the Junction Database. In such a case, to maintain self-consistency in our output data, the junction pair with the highest score is found – in the example either the pair of T-junctions or the pair of L-junctions – and return only that pair together with its score as output data.

If more than one junction in the current scan has a good match in the Junction Database, and the multi-junction match is good, then all well-matching junction pairs (with their matching scores) are passed as output data. Figure 30 summarizes the overall descriptor matching process. The single junction and multi-junction matching procedures are described in more detail in later sections.

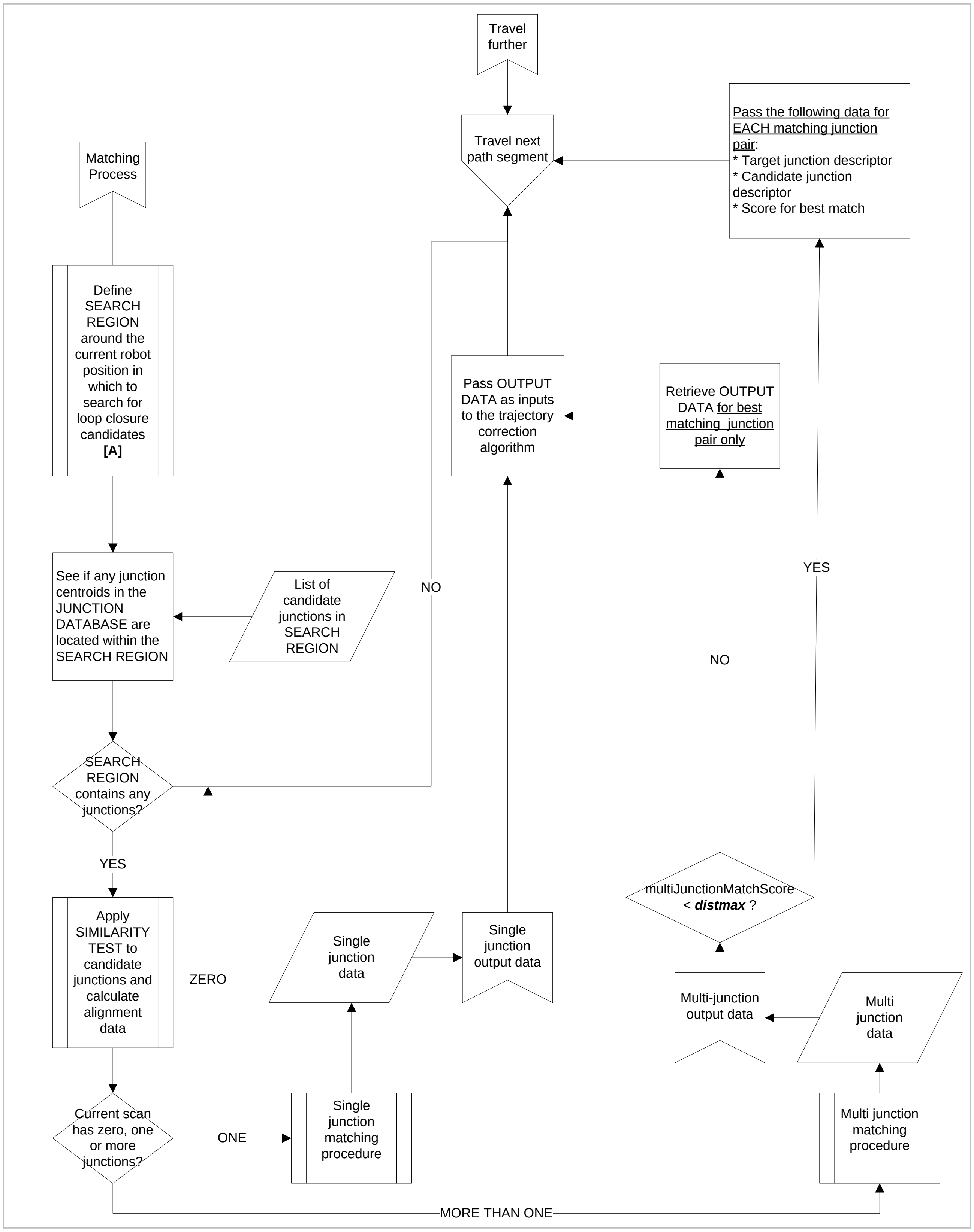

Travel further
Travel next path segment
Pass the following data for EACH matching junction pair:
* Target junction descriptor
* Candidate junction descriptor
* Score for best match
Matching Process
Define SEARCH REGION around the current robot position in which to search for loop closure candidates [A]
Pass OUTPUT DATA as inputs to the trajectory correction algorithm
Retrieve OUTPUT DATA for best matching junction pair only
See if any junction centroids in the JUNCTION DATABASE are located within the SEARCH REGION
List of candidate junctions in SEARCH REGION
NO
YES
NO
SEARCH REGION contains any junctions?
YES
multiJunctionMatchScore < distmax ?
Apply SIMILARITY TEST to candidate junctions and calculate alignment data
ZERO
Single junction data
Single junction output data
Multi-junction output data
Multi junction data
Current scan has zero, one or more junctions?
ONE
Single junction matching procedure
Multi junction matching procedure
MORE THAN ONE

Figure 30
Overall descriptor matching process

The similarity test for determining how many junction matches are present is now described. For each target junction in the current scan, each of the candidate junctions in the search region is compared with the target junction under consideration. The following procedure is used to evaluate how well each pair of junctions matches.

The first consideration is whether the two junctions have the same number of legs. This is very easily determined by comparing the number of elements in the two descriptors. If the two junction descriptors do not have the same number of legs, then this candidate junction is not considered further, and the next candidate junction is considered.

If the two junctions do have the same number of legs, then an orientation metric is calculated for the pair of junctions as follows. First, the translation between the two junction centroids is calculated so that the two junctions can be (conceptually) overlaid one on the other. Then, the listing of legs in each descriptor is reordered according to each leg's orientation angle (if the legs in the descriptors are not already ordered in this way). This ordering enables pair wise matching of the legs between the two descriptors.

The next step is to (conceptually) rotate the candidate junction about the overlaid centroids by increments of *deltaTheta*, a user-specified parameter, until a full 360-deg rotation has been made. Figure 31 illustrates an example of the overlay of two T-junctions, one of which has 90-deg angles, and one of which does not have 90-deg angles. At each increment of rotation, the squared difference between the angles of each pair of legs is calculated. Then, these squared differences are summed, and the sum is square rooted. This square root value is the score for that overlay angle. After all overlay angles have been considered, the maximum score is found, and that score is the metric that shows how well the two junctions match.

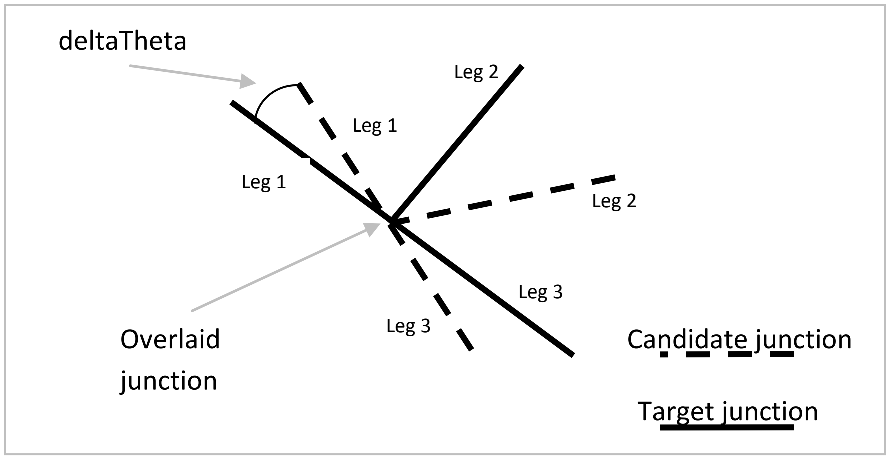


Figure 31
Overlaid junctions at a particular increment of rotation

Figure 32 summarizes the process for calculating the orientation metric.

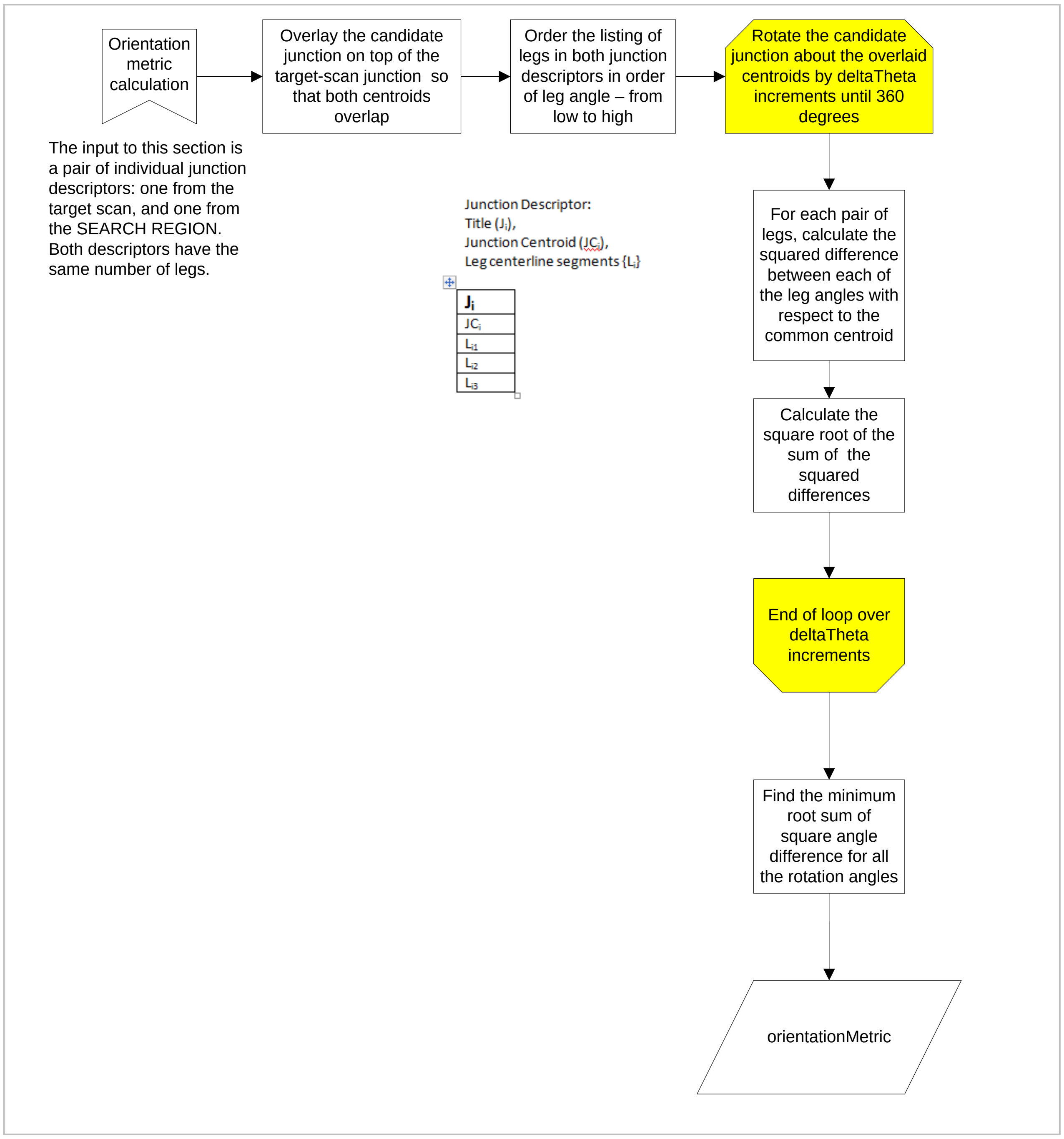

Orientation metric calculation
Overlay the candidate junction on top of the target-scan junction so that both centroids overlap
Order the listing of legs in both junction descriptors in order of leg angle – from low to high
Rotate the candidate junction about the overlaid centroids by deltaTheta increments until 360 degrees
The input to this section is a pair of individual junction descriptors: one from the target scan, and one from the SEARCH REGION. Both descriptors have the same number of legs.
Junction Descriptor:
Title (J_i),
Junction Centroid (JC_i),
Leg centerline segments {L_i}
J_i
JC_i
L_{i1}
L_{i2}
L_{i3}
For each pair of legs, calculate the squared difference between each of the leg angles with respect to the common centroid
Calculate the square root of the sum of the squared differences
End of loop over deltaTheta increments
Find the minimum root sum of square angle difference for all the rotation angles
orientationMetric

Figure 32
Orientation metric calculation

At this point in the algorithm, if a more complex junction matching metric is desired, it could be calculated here. Otherwise, processing is completed for this particular junction pair, and now, the next candidate junction is considered. After all of the candidate junctions for this target junction have been considered, the candidate match with the best matching score, *orientationMetric*, is identified.

The score is compared with a user-defined threshold, *angmax*, to determine whether this match is good enough to be considered a place recognition event. If the score is **greater than** *angmax*, then even the best matching candidate junction is not a close match to the target junction under consideration, and it is assumed that this target junction represents a place that has not yet been visited. If the score is **less than** *angmax*, then the candidate junction and target junction pair that achieved this match is preserved for further processing. However, first, any additional target junctions in the current scan are considered in the same way as was the first target junction. That is, the algorithm determines whether each additional target junction has a well matching (i.e., matching score < *angmax*) candidate junction in the Junction Database. The output of this whole junction comparison test is a set of target junction candidate junction pairs whose matching score is less than (i.e., better than) *angmax*.

The algorithm now determines whether there is zero, one, or more than one target junction that has a good match with some candidate junction. If there are no matches, then no further processing is performed for this pose in the trajectory, and the robot travels further. If there is one match, then the single junction output data (which has just been determined) is assembled and prepared for passing to an external trajectory adjustment program. The single junction output data is summarized in table 11.

Table 11
Output data for general junction based algorithm, single junction case

Target junction descriptor
Candidate junction descriptor
Matching score, *orientationMetric*

If there is more than one match, then the multi-junction matching process, described later, is now followed. Figure 33 summarizes the similarity test that has just been described for finding well-scoring matches between the set of all target junctions in a scan and the set of junctions in the Junction Database.

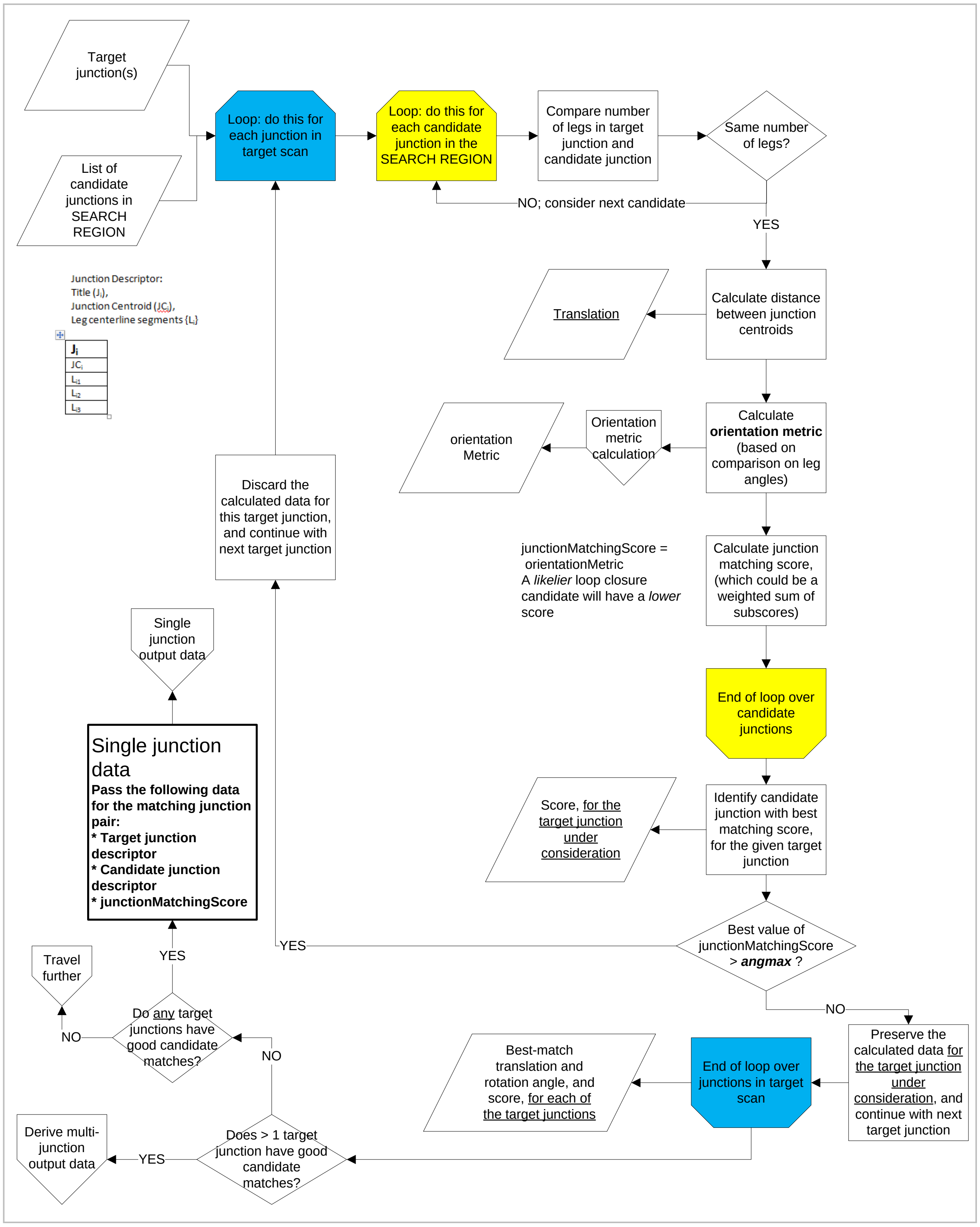

Target junction(s)
List of candidate junctions in SEARCH REGION
Loop: do this for each junction in target scan
Loop: do this for each candidate junction in the SEARCH REGION
Compare number of legs in target junction and candidate junction
Same number of legs?
NO; consider next candidate
YES
Junction Descriptor:
Title (J_i),
Junction Centroid (JC_i),
Leg centerline segments {L_i}
J_i
JC_i
L_{i1}
L_{i2}
L_{i3}
Translation
Calculate distance between junction centroids
orientation Metric
Orientation metric calculation
Calculate **orientation metric** (based on comparison on leg angles)
Discard the calculated data for this target junction, and continue with next target junction
junctionMatchingScore = orientationMetric
A *likelier* loop closure candidate will have a *lower* score
Calculate junction matching score, (which could be a weighted sum of subscores)
Single junction output data
End of loop over candidate junctions
Single junction data
Pass the following data for the matching junction pair:
*** Target junction descriptor**
*** Candidate junction descriptor**
*** junctionMatchingScore**
Score, for the target junction under consideration
Identify candidate junction with best matching score, for the given target junction
YES
Best value of junctionMatchingScore > ***angmax*** ?
Travel further
YES
NO
Do any target junctions have good candidate matches?
NO
Preserve the calculated data for the target junction under consideration, and continue with next target junction
Best-match translation and rotation angle, and score, for each of the target junctions
End of loop over junctions in target scan
Derive multi-junction output data
YES
Does > 1 target junction have good candidate matches?

Figure 33
Similarity test for target junctions of a particular scan

Multi-Junction Matching Procedure

The multi junction alignment process is now described. This process is reached if and only if more than one target junction in the current scan has a well-scoring candidate junction to match it. In this case, although each target junction has a good match, all of these matches must be self-consistent. The purpose of the multi-junction alignment process is to determine an overall matching error for the set of matches. This data enables a later part of the algorithm (fig. 30) to decide how to deal with a case in which the overall matching score is poor.

The input to the multi-junction matching process is the set of descriptors of target junctions, along with their well matching candidate junctions. The first step is to calculate the overall centroid C_J for the set of target junction centroids and the overall centroid C_D for the set of candidate thresholds. Figure 34 illustrates the calculation of the overall centroids, C_J and C_D.

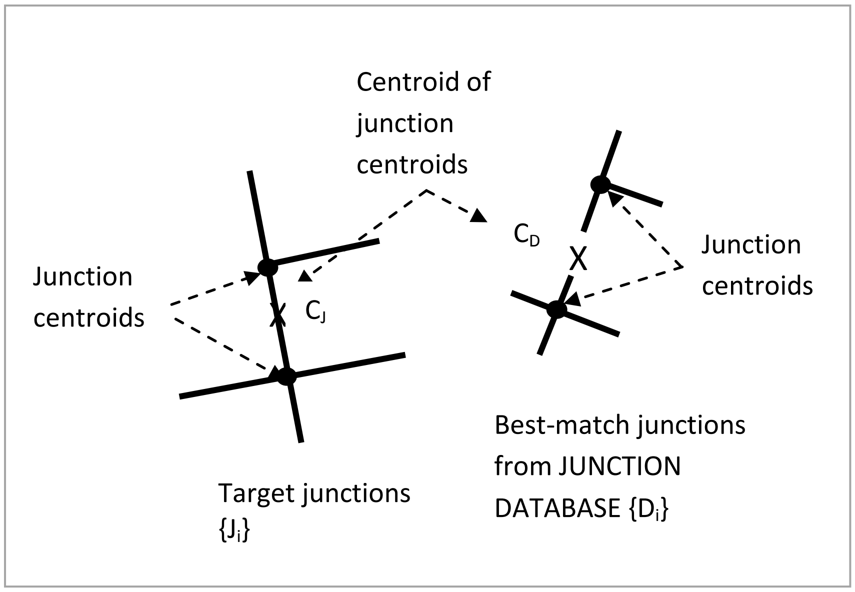

Centroid of junction centroids
C_D
Junction centroids
Junction centroids
C_J
Best-match junctions from JUNCTION DATABASE {D_i}
Target junctions {J_i}

Figure 34
Definition of overall centroids

Now, the 2D vector, $r = C_J - C_D$, is calculated. The entire set of candidate junctions is translated by the vector r so that the two centroids C_J and C_D overlap (conceptually). The remainder of the comparison process is similar to the process of comparing just one pair of junctions. Thus, the translated set of candidate junctions are rotated in increments of *deltaTheta* around the common centroid until a full 360-deg rotation has been made. The goal of this set of rotations is to determine whether the set of best-match candidates is self-consistent with the layout of junctions in the target scan. By doing these rotations, we find the rotational angle that gives the best match of the set of candidate scans to the set of target scans. This metric will be used to evaluate the self-consistency of the set of high-scoring candidate junctions.

For each increment of rotation, a distance measure (defined later) between each junction pair is calculated. Then, these distance measures are summed up over all the junction pairs that are being considered. To prepare for calculating the distance measure between a particular junction pair, each leg of the set of target-scan junctions is matched with a corresponding leg of the selected set of candidate scan junctions. This association is illustrated in figure 35.

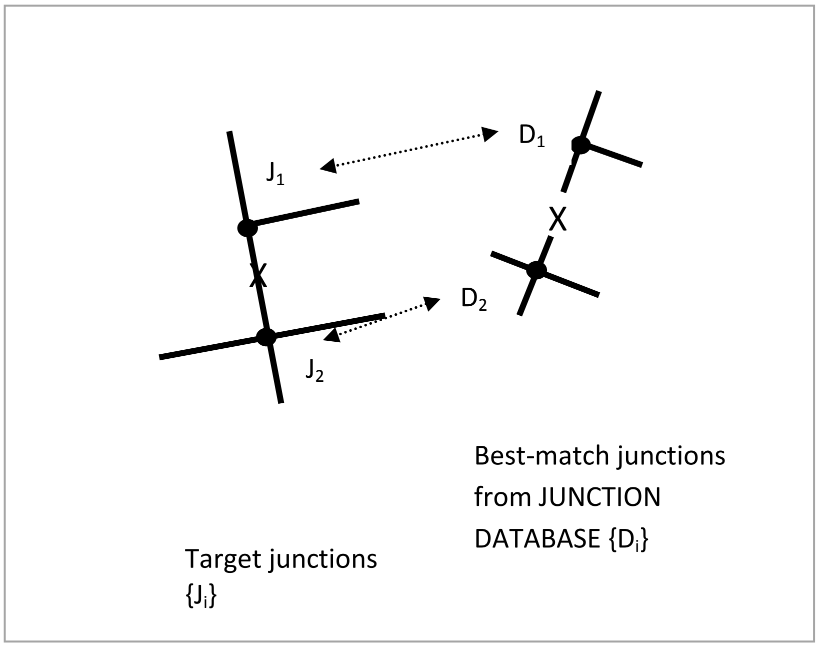


Figure 35
Association between target junctions and database junctions

The target junctions are denoted as J_1 and J_2, and the corresponding candidate junctions are denoted as D_1 and D_2. Figure 35 illustrates the matching process, which includes the junction pairs (J_1, D_1) and (J_2, D_2). Note that the two members of each pair were matched with each other earlier in this algorithm – the selected database junction was determined to be the best match with the target junction.

Now that each target junction is matched with its corresponding database junction, the individual legs from each junction must be matched with each other. To do this, the legs in each junction descriptor are ordered according to their angle, if this has not already been done. Then, the legs are matched pair-wise in a way that minimizes the total angle difference between all the leg pairs. This leg matching process is illustrated in figure 36.

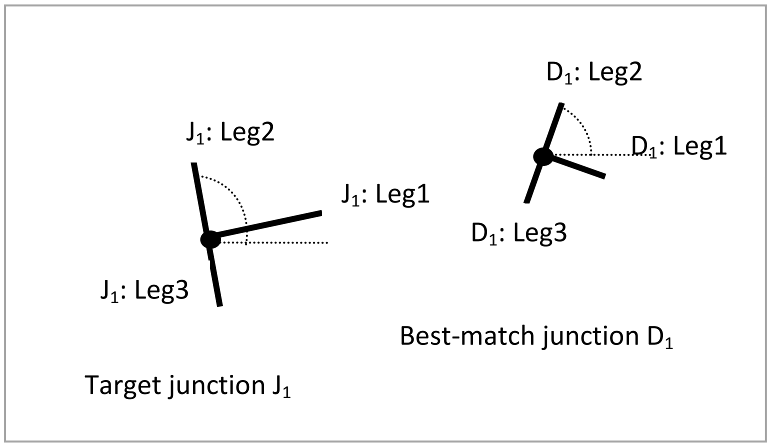


Figure 36
Leg matching process for two junctions

Figure 36 shows how, for each junction J_1 and D_1, the legs are ordered according to their angle with respect to the centroid of their own junction and a horizontal axis.

Now that each leg pair is defined, the distance measure between the two legs in each pair is calculated. Since each leg is a line segment, and the two line segments are in general not parallel, the distance between them is not well defined. This algorithm proposes a definition of distance between two line segments, and this definition will be used to calculate the overall similarity between the multiple junction pairs. Using the terminology shown in figure 37, the following procedure is used to calculate the distance measure between the two non parallel line segments, S1a – S1b and segment S2a to S2b.

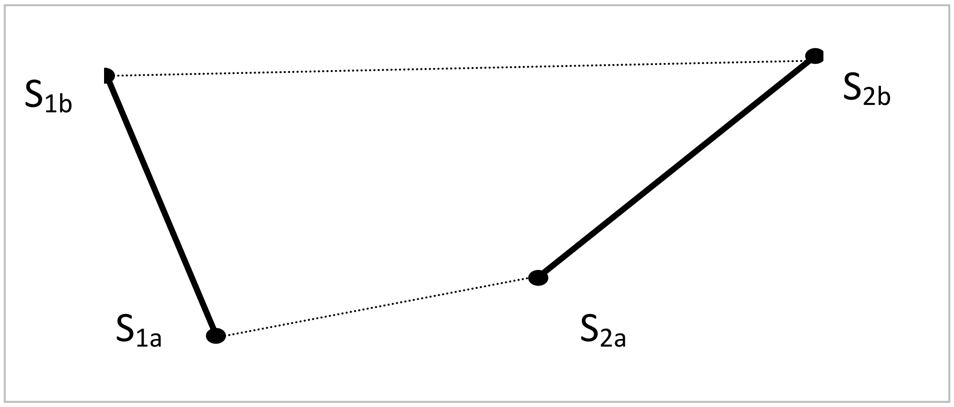


Figure 37
Terminology for calculation of distance between two line segments

- Let the nearest points of the two segments be called S_{1a} and S_{2a}
- Let the farthest points of the two segments be called S_{1b} and S_{2b}
- The non-parallel distance between the two segments is

$$\text{distance} = \sqrt{[\text{dist}(S_{1a}, S_{2a})]^2 + [\text{dist}(S_{1b}, S_{2b})]^2} \qquad (25)$$

where *dist* is the 2D euclidian distance function between the two points which are its arguments. It is possible to use other distance functions as well.

Now that the distance measure between each individual leg pair has been defined, the total distance measure between all leg pairs in the junction is calculated. This concludes the processing for a particular junction pair. This distance measure calculation is repeated for all junction pairs under consideration, and the sum total over all junctions is calculated.

Once the total distance measure between all junctions has been found for a particular increment of rotation, the next increment of rotation is considered, and this distance measure calculation is repeated for the new increment of rotation. After an entire 360 deg worth of rotations has been considered, the angle with the lowest distance measure is found, and the overall matching score between the two sets of junctions is simply that distance measure. This matching score is called *multiJunctionMatchingScore*.

This completes the calculations for the multi junction scenario, in which the target scan has more than one junction in it, **and** each of these target scans has a well-matching candidate junction in the Junction Database. The multi junction output data is summarized in table 12.

Table 12
Output data for general junction based algorithm, multi junction case

Target junction descriptor
Candidate junction descriptor
Matching score, *multiJunctionMatchingScore*

Figure 38 summarizes the similarity test that has just been described for finding well-scoring matches between the set of all target junctions in a scan and the set of junctions in the Junction Database.

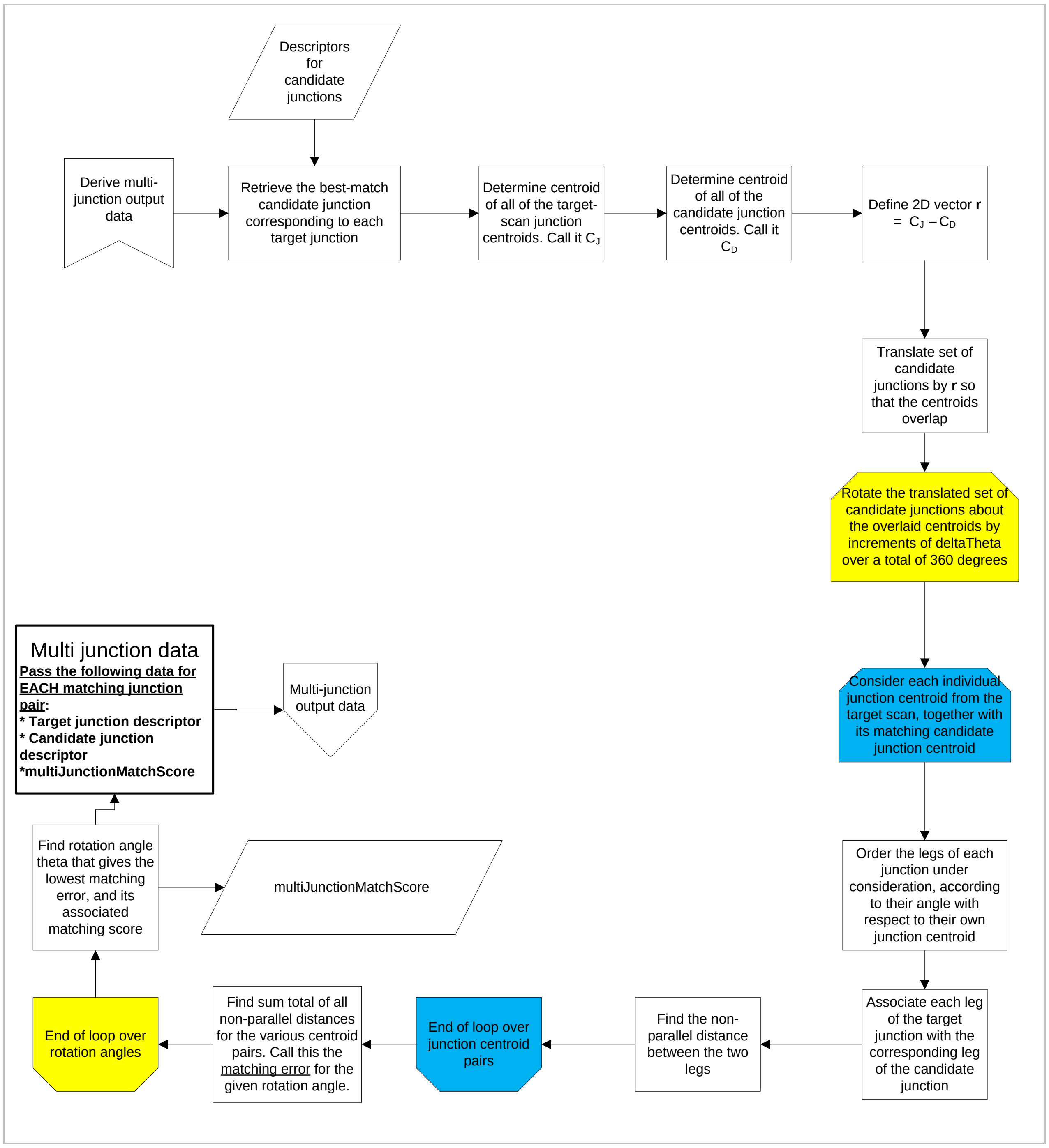

Descriptors for candidate junctions
Derive multi-junction output data
Retrieve the best-match candidate junction corresponding to each target junction
Determine centroid of all of the target-scan junction centroids. Call it C_J
Determine centroid of all of the candidate junction centroids. Call it C_D
Define 2D vector r = C_J – C_D
Translate set of candidate junctions by r so that the centroids overlap
Rotate the translated set of candidate junctions about the overlaid centroids by increments of deltaTheta over a total of 360 degrees
Consider each individual junction centroid from the target scan, together with its matching candidate junction centroid
Order the legs of each junction under consideration, according to their angle with respect to their own junction centroid
Associate each leg of the target junction with the corresponding leg of the candidate junction
Find the non-parallel distance between the two legs
End of loop over junction centroid pairs
Find sum total of all non-parallel distances for the various centroid pairs. Call this the matching error for the given rotation angle.
End of loop over rotation angles
Find rotation angle theta that gives the lowest matching error, and its associated matching score
multiJunctionMatchScore
Multi junction data
Pass the following data for EACH matching junction pair:
* Target junction descriptor
* Candidate junction descriptor
*multiJunctionMatchScore
Multi-junction output data

Figure 38
Calculation of multi-junction output data

At this point in the algorithm, it was shown how to calculate output data for two different scenarios: a single junction, or more than one junction, if the target scan has a well matching junction(s) in the Junction Database. The execution of all of the steps in figure 33, “similarity test for target junctions of a particular scan,” has been described. In the single junction scenario, the output data (table 11) is passed to an external trajectory correcting program, and the robot travels further.

In the multi-junction scenario, a bit of further processing is required. This processing is shown in figure 30. At this point, *multiJunctionMatchingScore* is compared with a user-specified threshold value, *distmax*, to determine whether the total set of target junctions has a good alignment with the matching set of candidate junctions. If *multiJunctionMatchingScore* is less than (i.e., better than) *distmax*, then all junction pairs with their matching scores are passed to an external trajectory correcting program. Otherwise, if *multiJunctionMatchingScore* is greater than *distmax*, then, as described earlier (see fig. 29 and accompanying discussion), only the best matching pair and its matching score are passed to an external program. In either case, once the output data is passed to the external program, that completes the processing for this step, and the robot travels further.

Overall Summary of General Junction Based Algorithm

Table 13 summarizes the user-selectable parameters for the general junction based algorithm.

Table 13
List of selectable parameters for general junction based algorithm

Parameter description	Variable name
Maximum distance between two line segments to be considered a hallway	*hallWidthThreshold*
Maximum distance between centerline endpoints for them to be considered part of a cluster	*clusterThreshold*
Minimum distance for search region	d_{min}
Maximum distance for search region	d_{max}
Increment of rotation angles when testing overlay of two junctions	*deltaTheta*
Threshold score for place recognition candidates	*angmax*

RESULTS AND DISCUSSION

This report presents three algorithms that can be used for automated place recognition in featureless, walled environments. It presents a new type of environmental descriptor that can be used to characterize featureless, walled environments such as featureless tunnels and building hallways. They could also be used in feature rich environments to add an additional method for confirming the validity of a place recognition decision.

The report also presents a descriptor matching process for the new descriptors. The algorithms include an approach for descriptor comparisons that is tailored to the new descriptors. They give as their output data the newly measured descriptor, the previously measured descriptor which has been identified as a match to the current descriptor, and a matching score that tells how closely matched the two descriptors are. This output can be passed to an external program that creates a map of the environment in real time.

The pair of descriptors that is passed to the external program constitutes a constraint in the network of nodes that represent each visited place. The matching score represents the strength of this constraint.

Each of the three algorithms has its own advantages and specific capabilities. The trajectory based algorithm is the simplest of the three – it uses only IMU data and does not depend on LIDAR data. This is both a strength and a weakness. It is a strength in that it could be implemented on a computationally constrained platform, perhaps even a smartphone, since the data it uses is very simple. On the other hand, it is a weakness since the algorithm does not take full advantage of the available data in case the robot has additional sensors on it such as LIDAR. It relies only on the trajectory, and implicitly on a walled structure that constrains the robot’s motion, to extract information about the environment. This is not, in general, a very robust approach to place recognition.

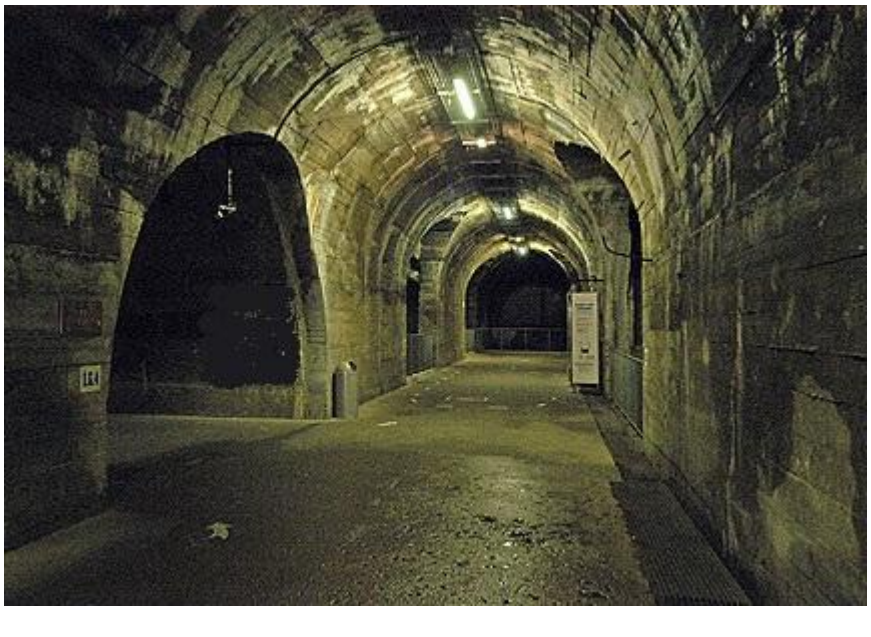
These weaknesses prompted development of the next algorithm, the LT-junction based algorithm. This algorithm uses distance-to-nearest-wall data as measured by LIDAR or some other ranging device. This usage of wall data makes the algorithm more robust than the trajectory based algorithm. However, the algorithm detects only L- and T-junctions, and by extension to +-junctions, which can be represented as a pair of back-to-back T-junctions. It cannot detect and form descriptors for more complex junctions, such as star shaped junctions.

The general junction based algorithm is a more robust algorithm than the LT-junction based algorithm. It uses mature and efficient image processing methods to extract data about even complex junctions. Furthermore, it presents a new type of junction comparison method – it allows multiple junction pairs to be compared both individually and simultaneously as a group. The group comparison method allows checking multiple junction pairs for self-consistency, which adds robustness to the algorithm. However, all this added robustness comes at a computational cost.

Consideration of this set of three approaches allows the user to select an operating point in the trade space of computational simplicity versus robustness.

CONCLUSIONS

The three algorithms presented here represent a set of three approaches to place recognition in featureless, walled environments, which is a very challenging type of environment for place recognition. This difficulty is easy to conceptualize by imagining oneself in a featureless tunnel network such as the one shown in the following figure.



Example of a featureless tunnel environment

It would be very easy for a human to become disoriented and lost while walking in such an environment. Robots navigating in such environments have similar difficulties, and they are technically difficult to address. Many types of feature descriptors have been presented in the technical literature, but environments like the one shown in the figure have very few unique instantiations of such descriptors.

The trajectory and junction based descriptors presented here, and the methods for comparing them, enable place recognition even in feature poor environments such as that shown in the figure. Thus, these descriptors and methods expand the frontiers of what type of environments can be accurately mapped by robots with inertial measurement units and possibly light detection and ranging (LIDARS). When used together with a place recognition system for featured environments, these algorithms can supply the missing piece for an all-environment mapping system.

RECOMMENDATIONS

Since these algorithms have not been implemented as of the date of this report, the author recommends implementing the algorithms in code, finding optimal values of the user-specified parameters, and testing their performance to verify and quantify their strengths and weaknesses.

Regarding the computational platform in which the algorithm will reside, the author recommends implementing the trajectory based algorithm in a smartphone or other such computationally constrained platform to test its effectiveness with such constrained resources. The LT-junction based algorithm and the general junction based algorithm are expected to require more computational resources, so they could be implemented in a computer that is integrated into a robot.

Once the algorithms have been thoroughly characterized via experiments, the author recommends integrating one or more of them into an external mapping program that adjusts the trajectory in real time based on place recognition events.

REFERENCES

1. Fallon M.F., Johansson H., Brookshire J, Teller S., and Leonard J.J., “Sensor Fusion for Flexible Human-Portable Building-Scale Mapping,” Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 4405-4412, 2012.

2. Cummins M. and Newman P., “Appearance-Only SLAM at Large Scale with FAB-MAP 2.0,” International Journal of Robotics Research, vol. 30, No. 9, pp 1100-1123, 2011.

3. Steder B., Ruhnke M., Grzonka S., and Burgard W., “Place Recognition in 3D Scans Using a Combination of Bag of Words and Point Feature based Relative Pose Estimation,” Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 1249-1255, September 2011.

4. Turner E. and Zakhor A., “Floor Plan Generation and Room Labeling of Indoor Environments from Laser Range Data,” Presented at International Conference on Computer Graphics Theory and Applications 2014, Lisbon, Portugal, January 2014.

5. Vallivaara I., Haverinen J., Kemppainen A., and Roning J., “Simultaneous Localization and Mapping Using Ambient Magnetic Field,” Proceedings of the 2010 IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), pp 14-19, Sept. 2010.

6. Zhang H., Martin F., “Robotic Mapping Assisted by Local Magnetic Field Anomalies,” Proceedings on the 2011 IEEE Conference on Technologies for Practical Robot Applications (TePRA), pp 25-30, April 2011.

7. Frassl M., Angermann M., Lichtenstern M., Robertson P., Julian B., and Doniec M., “Magnetic Maps of Indoor Environments for Precise Localization of Legged and Non-legged Locomotion,” Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 913 – 920, November 2013.

8. Dubbelman G. and Browning B., “Closed-form Online Pose-chain SLAM,” Proceedings of IEEE International Conference on Robotics and Automation, May 2013.

9. Bay H., Ess A., Tuytelaars T., and Van Gool L., "SURF: Speeded Up Robust Features," Computer Vision and Image Understanding (CVIU), vol. 110, No. 3, pp 346--359, 2008.

10. Fiolka T., Stuckler J., Klein D., Schulz D., and Behnke S., “SURE: Surface Entropy for Distinctive 3D Features,” Spatial Cognition III, Proceedings of International Conference on Spatial Cognition 2012, Lecture Notes in Computer Science vol. 7463, pp 74-93, August September 2012.

11. Stoyanov T., Magnusson M., Andreasson H., and Lilienthal A., “Fast and Accurate Scan Registration through Minimization of the Distance between Compact 3D NDT Representations,” International Journal of Robotics Research, vol. 31, No. 12 pp 1377-1393, 2012.

12. Tombari F. Salti S., Di Stefano L., “Unique Signatures of Histograms for Local Surface Description,” Proceedings of the 11th European Conference on Computer Vision: Part III, pp 356-369, 2010.

REFERENCES
(continued)

13. Nieto J.I., Agamennoni G., Vidal-Calleja T., "Loop-closure Candidate Selection by Exploiting Structure in Vehicle Trajectory," Proceedings on IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 92-97, 2011.

14. Point Cloud Library (www.pointclouds.org)

15. Open CV (www.opencv.org)

16. http://en.wikipedia.org/wiki/Gaussian_blur

17. http://en.wikipedia.org/wiki/Corner_detection

18. Zeng Z.Y., Jiang Z.Q., Chen Q., and He P.F., "An Improved Corner Detection Algorithm Based on Harris," Advanced Engineering Forum, vol. 6-7, pp. 717-721, 2012

19. http://en.wikipedia.org/wiki/Canny_edge_detector

20. http://en.wikipedia.org/wiki/Randomized_Hough_transform

LISTS OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

2D	two-dimensional
GNSS	global navigation satellite system
GPS	global positioning system
IMU	inertial measurement unit
LIDAR	light detection and ranging
RANSAC	random sample consensus
SLAM	simultaneous localization and mapping
SUSAN	smallest univalue segment assimilating nucleus

DISTRIBUTION LIST

U.S. Army ARDEC
ATTN: RDAR-EIK
RDAR-GC
RDAR-WSH-N, K. Patel
Picatinny Arsenal, NJ 07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste 0944
Fort Belvoir, VA 22060-6218

U.S. Army CERDEC
ATTN: Naomi Zirkind
Electronics Engineer
RDECOM CERDEC I2WD
(RDER-IWR-TE)
6003 Combat Drive
Aberdeen Proving Ground, MD 21005

U.S. Army TARDEC
ATTN: Lonnie Freiburger
6501 East 11 Mile Road
Warren, MI 48397

REVIEW AND APPROVAL OF ARDEC TECHNICAL REPORTS

Algorithmic Approaches for Place Recognition in Featureless, Walled Environments	
Title	Date received by LCSD
Naomi Zirkind	
Author/Project Engineer	Report number (to be assigned by LCSD)
2538 95	RDAR-WSH-N
Extension Building	Author's/Project Engineers Office (Division, Laboratory, Symbol)

PART 1. **Must be signed before the report can be edited.**

a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.

b. Use Distribution Statement A ✓, B____, C____, D____, E____, F____ or X____ for the reason checked on the continuation of this form. Reason: ____________

1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.

2. If Statement B, C, D, E, F, or X is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.

c. The distribution list for this report has been reviewed for accuracy and completeness.

George Papanagopoulos [illegible] 4-3-14

Division Chief (Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

George Papanagopoulos [illegible] 5-5-14

Division Chief (Date)

Stephen Leong [illegible] 5/6/14

RDAR-CIS (Date)

LCSD 49 supersedes SMCAR Form 49, 20 Dec 06