1. Introduction

There is currently considerable research interest in investigating the potential applications of groups of small, cheap, miniature robots [1]. Such robots are typically limited in their sensing ability and processing power. Much of the literature addressing how robots can construct maps of their environment is oriented towards robots with vision sensors, and relies on computationally expensive image processing algorithms. This paper examines how a miniature robot can build and maintain a useful map of its environment with short-range proximity sensors only, and do so in real-time even with the restricted processing power of such robots. This is a desirable ability because it could allow robots to be used in a range of niches where sophisticated sensing and image processing would be impractical. The novel techniques described in this paper have been implemented on a Khepera [2] miniature robot.

2. The importance of landmarks

It is useful for an autonomous robot to maintain a map of its environment for a number of reasons, such as planning efficient routes and avoiding cyclic behaviour. But the most fundamental use of maps is to prevent the robot losing track of its position relative to the rest of the environment. While a reasonable estimate of the robot's position can be maintained by tracking its movements and integrating them, unavoidable errors in the feedback from the robot's motors, and environmental interference, will accumulate over time to make these estimates increasingly inaccurate. The robot needs some way to compensate for these errors. This can be done by searching for "landmark" features of the environment that can be used as reference points. By detecting such landmarks in the environment and comparing them with the map, the robot can deduce corrections to its position estimate and keep better track of where it is relative to its environment.

3. Detecting landmarks

A robot with proximity sensors only cannot simply "look" at an object and recognise it. It can only sense the small portion of an object that is in its immediate locality. And even that portion may not be sensed very accurately. For example, proximity sensors may give a distance reading that is non-linear, noisy, influenced by ambient light, and the colour, texture, and other features of the object. Such readings are not even remotely suitable for direct use in landmark detection. There are simply no stable features the robot can recognise.

However, such non-ideal sensor data *is* sufficient to allow the robot to perform a simple task such as following the boundary of an object. To do this, the robot never needs to know the exact distance it is from the object, only whether that distance is increasing or decreasing. This *relative* information can be extracted reasonably reliably from proximity readings, even though the *absolute* distances the readings represent cannot. The reason this is relevant to a discussion of landmarks is that when the robot is following the edge of an object, the path it moves along will trace the outline of that object's boundary. The distance the path is from the object will depend on the exact nature of the object's surface- but whatever the distance is, it will be consistent, since the nature of surfaces tends to be remain constant. So if the robot follows the same boundary twice, the path it follows will generally be consistent. Hence recognisable features such as corners and edges that appear in the path will reappear in the same places the next time the robot follows the boundary. Therefore these features can act as landmarks for the robot.

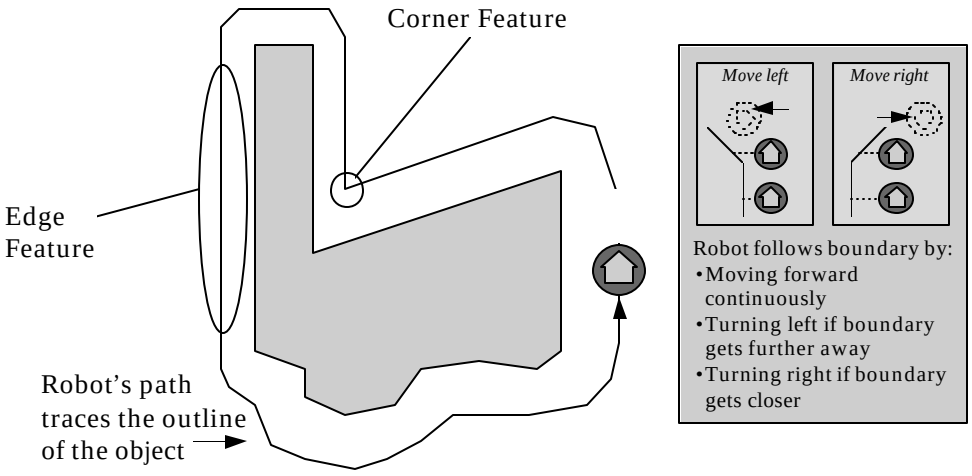

Figure 3-1: Tracing the outline of an object

This idea allows a robot to detect landmarks in its environment with even the most primitive sensing equipment. It could, for example, be implemented with a simple pair of "whisker" sensors- much cheaper than a machine vision system.

4. General strategy for trusting landmarks

There are two sections of a boundary which are particularly well suited to act as landmarks- corners and straight edge sections. These segments of the boundary have well-defined, stable features from which consistent information can be extracted, as will be shown soon. With careful use of this information, it is possible to compensate for accumulating error in the robot's estimates of its position and direction. Before looking at the details of how this can be done, some general difficulties with landmarks need to be addressed.

While the idea of landmarks is that they provide reference points that the robot can use to keep track of its position, it is important to remember that the robot's environment is not static. It cannot be assumed that a change in the apparent position of a landmark automatically means that the robot's estimate of its position has become inaccurate- it may equally well be that the landmark has simply moved. A change in the position of a landmark and an error in the robot's estimate of its position are indistinguishable while the robot is close to that landmark. However, an important point to realise is that they *can* be distinguished once the robot moves away from that area. A change in the position of a landmark will not change the position of any other landmark, whereas a drift in the robot's sense of position will change the perceived position of *every* landmark. One is a local change, the other is a global change. So if the robot finds that every landmark it meets seems to have moved, then it becomes more and more likely that the movement is only apparent, caused by a drift in the robot's estimate of its position. Essentially, the "consensus" of the landmarks in the environment is used to determine what has occurred in a particular part of that environment. This use of landmarks ensures that the map is kept self-consistent by essentially averaging error over the entire map[1].

[1] Note that this averaging process, while it keeps the map self-consistent, does not prevent the coordinate system of the map drifting over time. This point has no bearing on landmark recognition, but does have bearing on the nature of the services the map can provide.

A strategy for applying corrections to the robot's estimates is now presented that follows from the above considerations. It is first assumed *for the purposes of calculation* that apparent changes in the environment are entirely due to drifts in the robot's position and direction estimates and not to changes in the position of landmarks. Then, having computed the corrections to the robot's estimate of its position and direction that would be appropriate under that condition, only a *conservative fraction* of each correction is actually applied. If the apparent changes in the environment are in fact due to drifts in the robot's estimates, then every landmark the robot meets will continue to apply these corrections until they build up sufficiently to compensate for the drifts. If on the other hand the apparent changes are real, and due to an actual change in the position of a landmark, the correction will not be reinforced at any other landmark. In fact, at other landmarks the "incorrect correction" will be compensated for, since the error introduced into the robot's estimates will seem just like a natural drift from accumulating error and can be corrected as such. The "conservative fraction" mentioned earlier should be one chosen so that if a correction is applied in error, it is small enough to be recovered from through the same process that deals with normal drift. Suitable values will be quoted for straight-edge and corner landmarks when they are discussed. It is important to note that corrections to the robot's estimate of its *direction* must be made particularly conservatively, because small erroneous corrections to the direction will be multiplied into very large errors in its position as the robot moves. Erroneous corrections to the position estimate do not become amplified in this way[2].

This ability of the landmark system to "heal itself" if errors are introduced while attempting to make corrections is quite general, so long as the errors are not large enough to prevent landmarks being recognised. One useful consequence of this is that it is acceptable to make approximations when calculating the corrections to the robot's estimates from a landmark. The difference between the approximation and the exact answer can be seen as introducing an error component into the corrections the robot makes which may add to the drifts in the estimates rather than removing them. This will simply appear as an extra component in the position and direction drift calculated in future corrections and be eliminated.

Before looking at the details of detecting particular landmarks, it is necessary to choose how the robot's map is to be represented. This is discussed in the following section.

5. Representation scheme

The two most widely used map representation schemes in robotics are grid-based and topological [[1]]. A grid-based scheme assumes the world is arranged in a manner somewhat analogous to a chess board, with all the objects in it arranged in definite locations that can be known exactly by the robot. If the robot can know its own position precisely, and is able to sense objects perfectly, there is no problem with this, but otherwise the scheme becomes essentially unworkable. It is very difficult to allow for any uncertainty in the robot's position in it, and limitations in the robot's sensing capability are particularly troublesome. Hence it is unsuitable for use with an autonomous robot, although it may work perfectly well in a simulated environment.

Topological schemes, in contrast, depend less on the exact location of the objects in the robot's environment, and concentrate more on trying to capture the basic shape of the environment in terms of its essential topology. In other words, the robot tries to determine which areas can be reached from each

[2] If you aim a cannon at a target, but shoot a few degrees in the wrong direction, you will miss the target by a distance that keeps increasing the further the cannonball goes. If, on the other hand, you aim the cannon exactly on target, and move it a few paces left or right before firing- still facing the same direction- then the cannonball will miss the target by just the distance you moved no matter how far away the target is.

other, which are cut off from each other by obstructions, etc. As such the precise details of the robot's surroundings are not as significant, only the overall form, so uncertainty in the robot's position becomes less important. However the price paid for this is the extra computation necessary for recognising abstract topological features in the environment. For a robot with no long-range sensors, such features cannot be directly observed. The only way to observe them is indirectly, by keeping track of the discernible attributes of the immediate environment, combining that information into a model of the overall environment, and then analysing that model for topological features. In other words, in the absence of long-range sensors it is effectively necessary to build a map before topological features can be detected, so such features cannot be used in building the map in the first place.

A more suitable map representation scheme can be derived if, instead of seeing a map as a model of the environment, it is viewed more as a "record of experiences". At any particular moment, the robot experiences the environment in its immediate vicinity, as perceived through the robot's sensors. It also experiences feedback from its motors indicating any movement it is making, which can be used to calculate its position. It is clear the most well informed robot conceivable is one which archives all this information exhaustively, continuously recording the state of the sensors along with the associated position of the robot, and never discarding any of this data. Such a robot is an upper bound on the knowledge a cartographic system can have[3], since it contains every piece of data ever available to the robot. The approach used in this project takes this trivially simple idea of archiving all the robot's experiences as a starting point, and modifies it into a form that is actually practical to implement.

Firstly, a complete archive of sensor state and motion feedback for all the time the robot is in operation is obviously hopelessly impractical- the robot would quickly "drown" in the information deluge, with too much data to process. To reduce this burden, it would seem reasonable to eliminate any records in the archive that apply to the same position of the robot as a new record being added, since the new record will have the most up-to-date information about the state of the environment at that position. However there is a problem with this. The robot's position is deduced by integrating the motion feedback from the robot's motors. Errors in the motion feedback are unavoidable, so the position estimate will gradually drift from the robot's true position. Hence two records showing the same "nominal position" of the robot cannot be assumed to correspond to the same physical position if there is a significant time interval between when they were recorded. However, if there was some way to use landmarks in the environment to keep the robot's estimate of its position consistent, then it would be acceptable to keep only the most recent record corresponding to a given position and discard all previous records for that location.

At this point an initially *unjustified* assumption is made that this is indeed the case- that the cartographic system eventually constructed will be able to keep track of its position by using landmarks. Given that, we can discard as out-of-date any entries in the archive that are marked at the same position estimate as the current position when we add a new record. This step will be justified in Section 4, where it is shown that the representation scheme which the assumption made here leads to will in fact allow the robot to keep track of its position by the use of landmarks.

Another change to the archiving strategy needs to be made before it becomes practical. Obviously records of sensor state versus robot position cannot be made for every continuous point along the robot's path, as this would still demand unlimited storage space. Instead, it is acceptable to record "samples" at positions with a certain minimum distance between them- so the robot has information about representative points in every area it has been. The minimum distance between records is chosen based on how much memory is

[3] Of course, trying to make use of this huge archive of data sensibly would be another matter entirely.

available to store the samples- the smaller the distance, the more samples in a given area and the more memory that will be required.

At this point, the "record of experiences" has been re-cast in a form that is actually practical to implement. The records of sensor state versus nominal position are called *"markers"* in this paper. The following example clarifies the proposed nature of the "marker"-based representation scheme.

By using markers, the robot can deduce what its sensors were reading the last time it was in a particular position. But it is not as easy to work the other way, and use markers to estimate the robot's position from its current sensor readings. This is because there may be many markers with the same set of sensor readings associated with them, and the robot will not be able to distinguish them. The robot's sensors are not rich enough to record its environment at a level of detail that would make this distinction possible- for example, all the markers that are away from boundaries look the exact same to the robot: zero proximity readings on all sides. This is why it is important to try to isolate special "landmark" features in the environment that the robot *can* distinguish from other areas and use to keep track of its position. Although most markers of themselves cannot be used as landmarks, there are some markers- or *groups* of markers- that can.

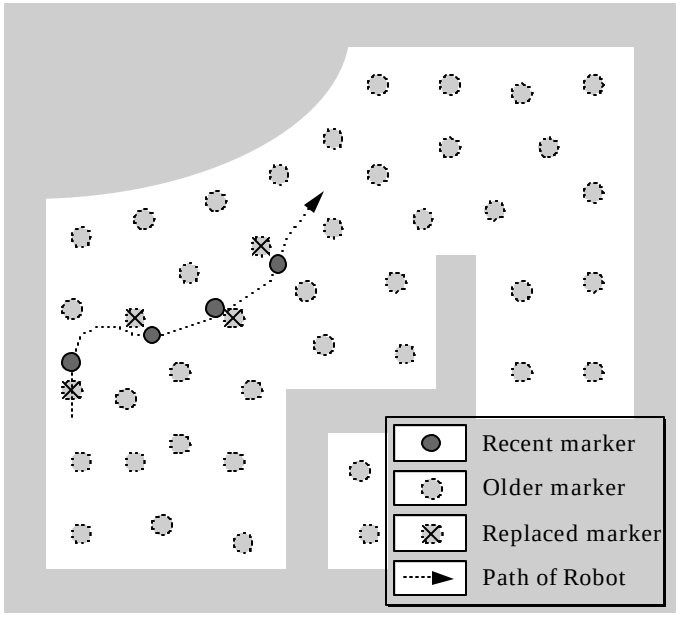

Figure 5-1: A map as a collection of markers

The robot's "map" in this form of representation is simply a collection of markers. Since the collection is potentially large, it is important that it be structured in a way that lets the robot extract the information it needs from the map in a timely fashion, without having to do any computationally expensive searching. This can be achieved by arranging markers into a hierarchy of "neighbourhoods"- structures for filtering groups of markers by their distance from the robot so that it can find out about its locality in a timely fashion without searching the entire archive. Each neighbourhood has an associated nominal distance range with it. The robot attempts to keep markers whose positions are within that distance range from the current robot position in the appropriate neighbourhood. The distance range associated with a neighbourhood is called *nominal* because in general the distance of some markers in a neighbourhood from the robot's current position will in fact lie outside the neighbourhood's specified distance range. This happens when a movement of a robot changes the distance from the robot to a marker sufficiently to make it inappropriate

for the neighbourhood it is in. The marker will be re-classified into the correct neighbourhood the next time it is examined by the sorting process.

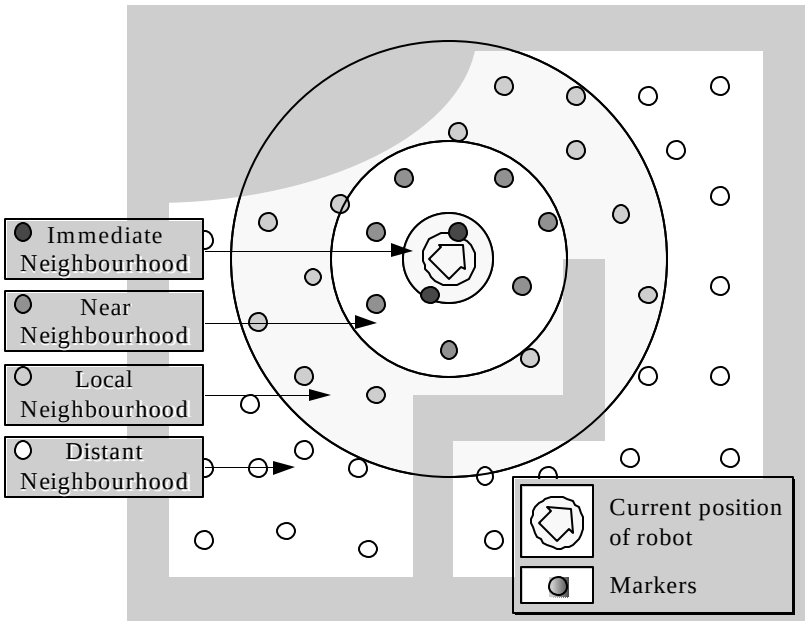

Figure 5-2: Distance ranges associated with neighbourhoods

6. Straight edge landmarks

The effect of a drift in the robot's position estimate is shown in Figure 6-1. The robot is shown following the same edge segment twice. As the robot follows the edge for the first time, it lays markers in a straight line at regular intervals along it. When the robot returns to the edge at a later stage, any drift in the robot's position estimate will cause the markers it lays this time around not to be collinear with the ones laid originally[4]. This effect can be measured and used to correct the robot's estimates of its position.

[4] Unless the drift happened to take place along the direction of the edge itself. This possibility will be discussed soon.

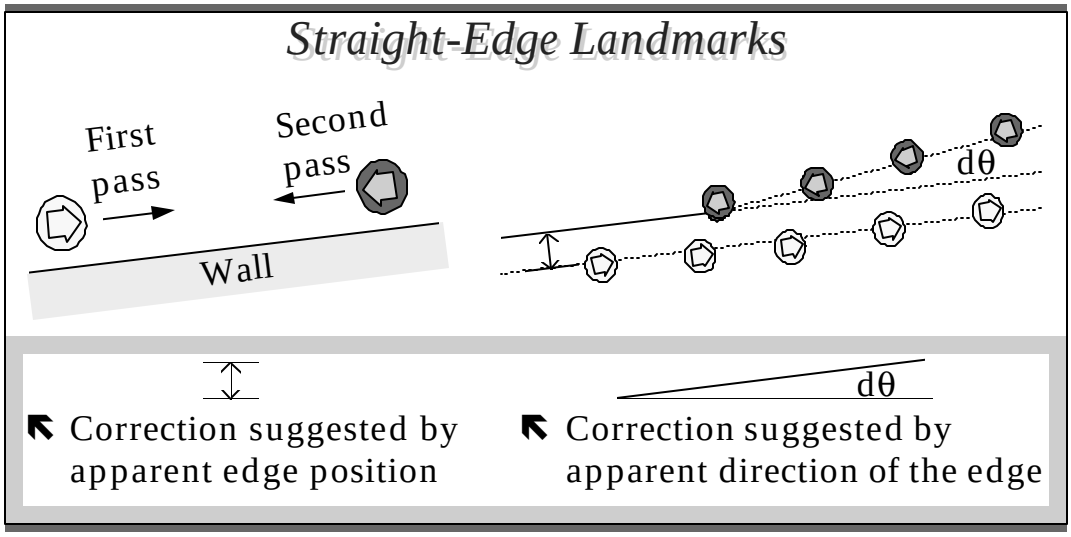

Figure 6-1: mumble mumble straighty

7. Corner landmarks

Consider the situation where the robot is following an idealised boundary as shown in Figure 7-1. The boundary is straight initially, then turns, then continues straight again in another direction. The robot's movement will reflect the shape of the boundary. By monitoring its motion, the robot can detect corners such as the one in the boundary shown here, and use them as landmarks. This section discusses how this can be accomplished.

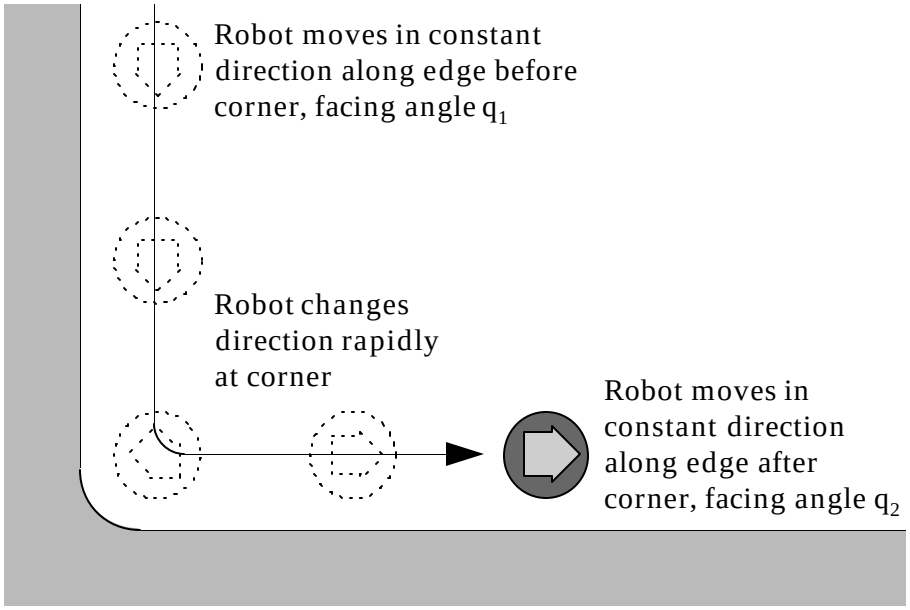

Figure 7-1: Robot moving around a concave corner

It is possible to estimate how sharply an edge is turning simply by measuring how quickly the direction of the robot is changing as it follows that edge. Therefore there is no difficulty in detecting and characterising

curves in the boundary. The problem is making use of that information. There is no special point along a curve that the robot can distinguish from all others and use as a landmark. However, the sharper a curve is, the less distance it can extend. A gentle curve can extend a great distance, but a sharp curve must end quickly or the boundary will turn back in on itself. If a curve is sharp enough, then the distance it extends will appear as a point to the robot. More specifically, if the distance a curve extends is close to the granularity at which the robot is mapping its environment, then that curve can be used to isolate a point in the environment that can be treated as a landmark.

For the robot used in this project, a "corner" was considered to be any curve that made the robot turn 30° or more in one second of motion. Such a curve appeared as a well-defined point. Note that concave corners are useful, convex ones are not. The reason is that the distance the robot has to travel to turn a convex corner cannot decrease lower than a fixed limit caused by the shape of the robot itself- see Figure 7-2.

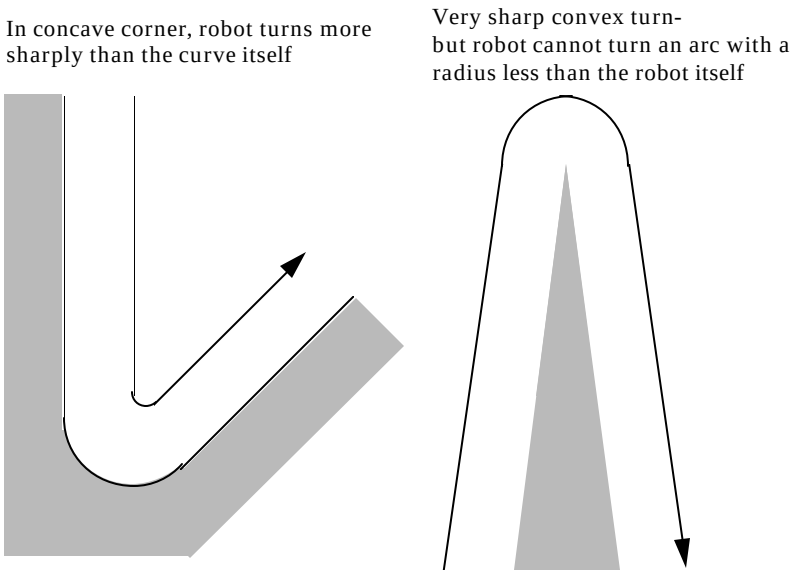

Figure 7-2: Concave versus convex turns

Once a corner has been recognised, it can be used to correct the robot's position estimate. Any difference between the position at which the robot turns the corner and the position of the marker laid at the corner by the robot the last time it turned it may be due to a change in the environment or to a drift in the robot's position estimate. It was decided to give the current and previous corner position equal weighting in calculating the corrected position of the robot, but any conservative ratio produced satisfactory results.

Note that corner landmarks cannot be used directly to correct the direction estimate. The directions of the robot before and after the corner are not known accurately enough for this. However, *pairs* of corners can be used indirectly to correct the direction estimate. Consider the situation shown in Figure 7-3. Here the robot has passed two corners. At the first corner it met, it found a discrepancy between its current position estimate (b_1) and the position at which the corner had been detected the last time the robot passed it (a_1). It used this discrepancy to compute a corrected position, c_1, that was simply the average of a_1 and b_1. At the next marker, it repeated the same process. However it can then be observed that when the robot passed these corners last, they were in the positions a_1 and a_2, but this time round they have appeared to

be in locations c_1 and b_2[5]. This indicates a direction drift of $\Delta\theta$ as shown, which can be applied as a correction (weighted conservatively, as always).

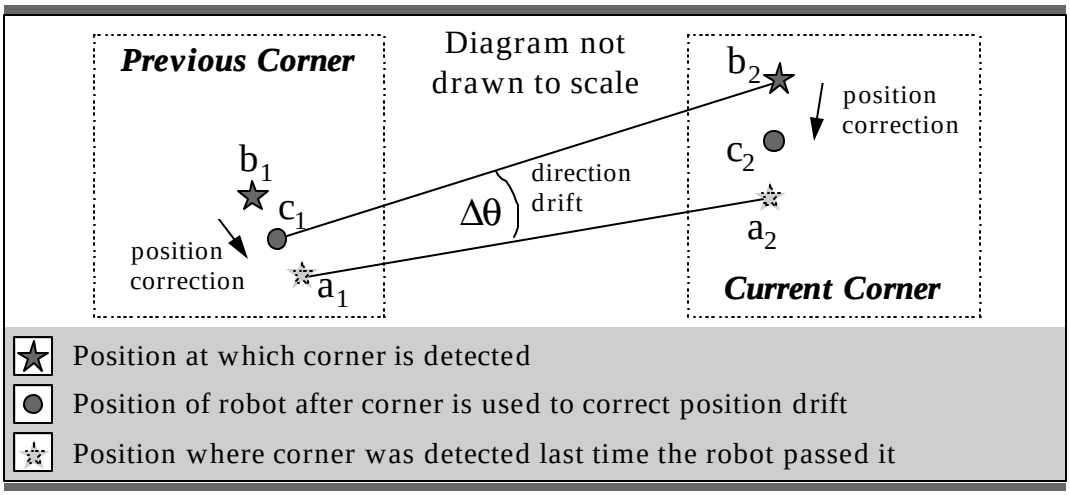

Figure 7-3: Sequential corner landmarks

The discussion of the use of landmarks is now complete. The two types of landmarks developed were found in experiments to work well, and they demonstrate that the marker representation scheme is in fact workable by showing that the robot can keep its position and direction estimates consistent relative to its environment. See Chapter 8 for experimental results that illustrate the operation of the landmark system in practice.

The use of landmarks concludes the examination of how the robot's map is built and maintained. The following sections look at how the map can actually be accessed by behaviours that need to use it.

8. Interacting with the map

The chapter so far has concentrated on the issue of building and maintaining a consistent map of the robot's environment. Equally important as building the map is how to go about actually making use of it. This section discusses how the cartographic system can provide useful services to modules that wish to interact with the map.

8.1 Virtual Sensors

Often modules do not need detailed qualitative information from the map, just answers to simple quantitative questions like "how familiar does the current location of the robot seem?" or "how confident is the robot of its position estimate?". These statistics can be provided in the form of "virtual sensors" that appear to behaviours just like physical sensors, but are internally generated. Some useful virtual sensors that can be provided are as follows:-

- **Familiarity**- this sensor indicates whether the robot is currently in a region that the cartographic system recognises, and if so how long has it been since it has last there. This gives the robot a sense for how

[5] b_2 is used in the calculation rather than c_2 because the position correction applied at the second corner actually obscures the direction drift by compensating for some of the position drift it caused as the robot moved from c_1 to b_2. Remember that as the robot moves, any error in its direction estimate is reflected as a growing position drift. To compute the error in the direction estimate, all the position drift should be taken into account.

recently it has passed through a particular area, or if the area is totally unknown to it. Familiarity is generated very simply from a comparison of the current time with the timestamp of the nearest marker to the robot which has not been laid recently, if one is present. This sensor results in a simple number, yet cannot be supported without the full effort of the cartographic system.

- **Confusion** - this sensor measures how uncertain the cartographic system is about the accuracy of its best guess at the robot's position. This uncertainty grows with the length of time the robot is moving after it has last managed to get a fix on its position from a landmark. This gives the robot a sense of how long it has been in motion without getting some fix on its location. Confusion is zeroed when the robot meets a corner landmark, or when two edge landmarks are passed with edges that are at an angle of at least 45° to each other (see Section 5, page 3).

8.2 Goal Seeking

It is also useful to have a service that uses the map as a resource to plan an efficient route to a given target location. This is a particularly difficult situation in which to avoid shared representation between the service and the user. First there is the problem of how to set the target without requiring the behaviour using the service to have access to the map. This is achieved by using the Tagging Service described above. A limitation of this is that targets may only be places the robot has passed through at some point in its history- but this is reasonable since the map would be of no use for planning routes to areas in which the robot has never been[6]. Another problem is how to communicate the route this service generates back to the behaviour that uses it without a shared representation of locations and paths. The solution adopted was to implement a virtual "*scent*" sensor. This sensor was generated in such a way that the robot could reach the target simply by moving in directions of increasing "scent", and moving away from directions in which the scent's intensity decreased. This is a simple reactive strategy, with no shared representation with the cartographic system needed. The "scent" sensor represents the gradient of the cartographic system's estimate of some cost function from the robot's current location to the target, but none of the complexity of calculating that function is visible to the user. This idea works out to be something quite similar to the "Internalised Plans" technique discussed in [HIT make coherent]. The details of how planning is achieved within the marker map representation scheme are now examined.

Markers are stored in a system of neighbourhoods designed to efficiently filter out which markers are close to the current position of the robot, since these are the most relevant to it for most purposes. However, in planning routes to arbitrary targets, information about markers distant from the robot's current position is needed. Specifically, it is important to be able to determine which markers can be reached from each other, and how great a distance the robot has to travel to do so. The use of neighbourhoods only allows the robot to determine which markers are close to the current position of the robot, and it cannot be used to determine which markers are close to some other arbitrary marker.

To solve this problem, extra "connectivity" data is added to markers. As the robot moves, the neighbourhood system determines which markers are close to its current position. When the robot lays a marker and moves away from it, that information about adjacent markers can be captured and stored in the marker as connectivity data. It is then possible for planning to be done at a later stage, using these "frozen images" of the information calculated by the neighbourhood system. Figure 8-1 shows an example of a collection of markers with connectivity information overlaid that would be suitable for planning routes with.

[6] This is true for autonomous robots, which are entirely responsible for generating their own map of the environment and hence cannot know anything about places they have never been. It would not apply if the robot had a built-in map of some form.

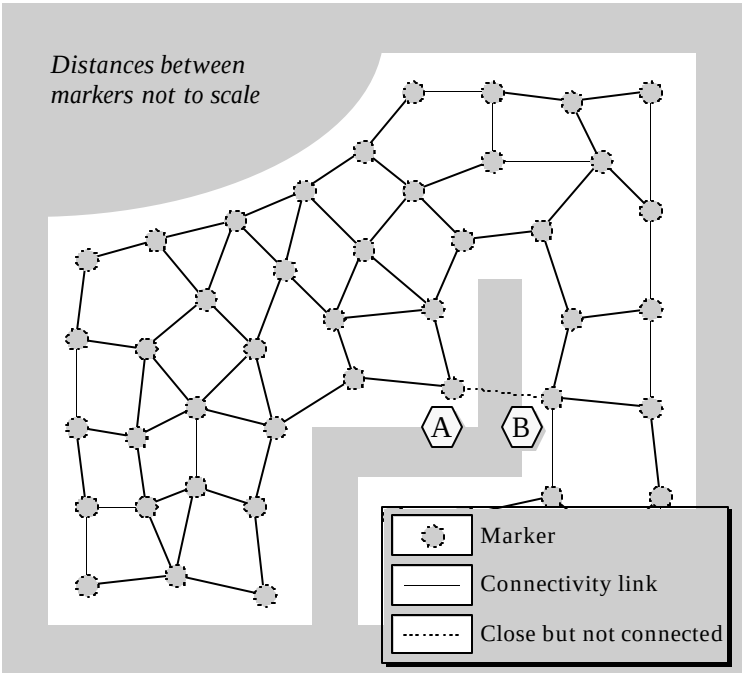

Figure 8-1: Storing connectivity data with markers

The diagram draws attention to the fact that the robot cannot assume that, just because two markers are close to each other, it is possible to move from one to the other- they could be on opposite sides of a wall, for example, as is the case with markers A and B in the diagram. Care must be taken to detect such conditions. Connectivity links should only lead from one marker to other nearby markers which the robot can move to *without hitting an obstruction*- otherwise they will be misleading and useless for planning routes with. Hence two markers being close to each other is a necessary condition for them to be considered connected, but it is not sufficient.

One sufficient reason for considering two markers to be connected is if they are laid one after the other by the robot. Such markers represent successive points along the path of the robot, so it is reasonable to assume that they can be reached from each other since the robot has actually just done so.

From the connectivity information this gives, the robot can deduce further appropriate connections to make. Two markers that are close to each other but were laid at different times can be deduced as being connected if markers close to one of them are connected to markers close to the other. .... Since the marker being laid and the marker being replaced represent the same physical area, connectivity data from the replaced marker can be transferred over to the new one (with some caution, as will be discussed in a moment).

Connectivity information is different to other data about the environment stored in markers because, by its very nature, it cannot be derived from purely local considerations. In contrast, information about whether an area is beside a boundary, for example, can be constructed entirely from immediate sensor data. It makes sense for the robot to always discard such data when it is replacing markers, and derive it anew for the current state of the environment. The robot should trust its sensors over any derived source of information it has, so the map should be overwritten with actual sensor data whenever possible. However, when working with connectivity data, information is lost if the old markers are simply discarded because connectivity information cannot be reconstructed completely from immediate sensor data. If the new marker being laid at C had replaced the older marker without copying its connectivity data, the robot

would no longer know that the area the markers represent is connected to D. Therefore, connectivity data should be transferred across to new markers from the markers they replace.

If the links are simply copied to the new markers, the system works reasonably well for a while. However the markers the robot lays are not constrained to be in the exact same location as the markers they replace- and in general they will not be. Hence as the robot moves back and forth across the same area, and the links are copied between successive replacements to the original marker, it is quite possible that the position of the marker holding the links may have drifting quite a distance from the position of the original marker. The links will then give a totally inaccurate picture of the reality.

To step around this problem every marker is given a "Reachable" timestamp. When the robot passes close enough to a marker to be confident that there is no boundary between the robot's current position and the marker, this timestamp is set to the current time. This it taken to indicate that this marker was reachable by the robot at the given time. The timestamp is actively spread to any markers that are linked to it (with some time subtracted to represent roughly how long it would take to get to that marker). This is taken to indicate that those markers could have been reached at the calculated time if the robot had chosen to do so. These timestamps continue to spread from marker to marker. Then, when the robot replaces markers, connectivity can be reconstructed by comparing "Reachability" times with other markers in the locality. If two markers are close in position and both could have been reached within a short time from the robot's current position (as indicated by them having reachability timestamps close to the current time), then those two markers can be considered reachable from each other and have their connectivity links updated appropriately. This is robust, and not subject to marker drift as simply copying connectivity data would be.

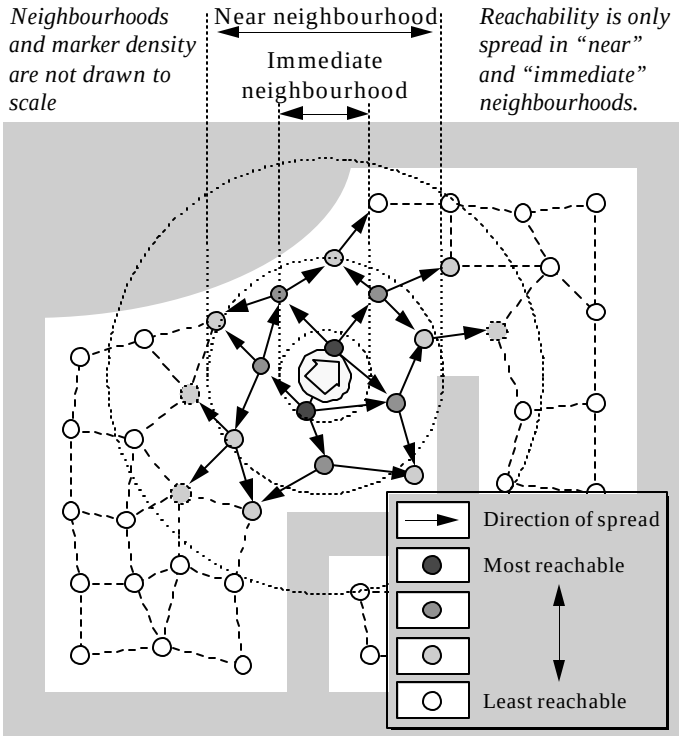

Figure 8-2: The use of reachability

To limit the computational burden on the robot, reachability is only spread within the immediate and near neighbourhood of the robot. Any markers outside of these neighbourhoods are not reachable from the robot's current position within a short time, so it is reasonable to eliminate them from consideration anyway.

There is one special case that needs to be catered for to ensure the correct operation of the reachability system. It is possible that reachability could spread to a small extent around a narrow wall.

However, it is simple to introduce a heuristic to stop connectivity creeping around a wall, by simply disallowing links to be made between close markers that are both on a boundary, with the boundaries facing in opposite directions away from each other- i.e. on opposite sides of a narrow wall.

There is one final practical consideration that is useful for storing connectivity information efficiently. It has been shown that if the robot is aware that two markers are linked, it may deduce that other markers are reachable from each other by spreading reachability. Hence not every link between markers needs be stored, only a sufficient number to allow reachability spreading to deduce the rest. Due to the memory limitations of the robot this work was implemented on, the number of links from each marker was limited to four. By applying criteria that favoured storing links to markers lying in different directions over links to markers clustered in the same direction, four links were found to be more than adequate for correct functioning of the reachability system- i.e. by spreading reachability, the robot successfully avoided losing connectivity information when it replaced old markers with fresh ones. The first diagram in this section, Figure 8-1 on page 11, in fact showed connectivity data constructed with a maximum of four links from each marker.

Given that connectivity links are being maintained, goal seeking is straightforward to achieve by any standard search technique. One simple way it can be done is to assign a "hop count" to every marker to represent how many other markers the robot would need to pass through when going from that marker to the target. Obviously the hop count of the target is zero. Any markers linked to the target will take on a hop count of one, and markers linked to them in turn will take a hop count of two, etc. In general, a marker M determines its hop count by finding which of its links leads to the marker with the lowest hop count. It should assign itself a hop count of one greater than that, and tag the link as shown in Figure 8-3.

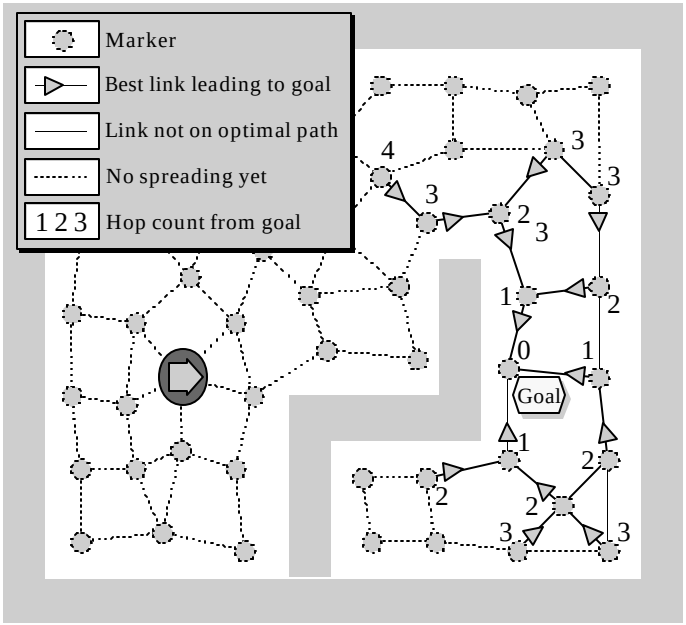

Figure 8-3: Goal Spreading in action

Once this process "spreads" out to markers in the vicinity of the robot, it can simply follow all the tagged links to the target. This is called "Goal Spreading". It can be improved by using cumulative distances between markers on the route to the goal instead of simple hop counts, or a combination of both (as was

used for the robot this work was implemented on). The computation required for Goal Spreading is extensive and takes time, although the work done for each marker is quite simple. The calculations should be performed as a background task so that they do not affect the real-time performance of the robot.

There is one final improvement that can be made to Goal Seeking. If *multiple goals* were set for the robot, everything described so far would still work- Goal Spreading would simply spread paths out from each goal, and whichever reached the robot first would be the one it would move towards. This is suitable for a situation where a number of goals are equally acceptable to the robot. It is simple to extend Goal Seeking so that it can also handle cases where some goals are more desirable than others. This is done by associating a "desirability" factor with the goals when they are set, and spreading that factor to markers that point along routes to that goal. In other words, when a marker scans the markers it is linked to for the one with the lowest hop-count/cost to a goal, it should only consider those with the highest "desirability" present, and then accept that desirability level for itself. Hence as paths to more desirable goals propagate, they can "take over" markers that were leading to less desirable goals- even if those goals were closer.

The results of goal spreading are made available through the "scent" virtual sensor, which simply gives a vector corresponding to the direction of the tagged link of the marker nearest the robot's position, if goal spreading has reached that link. The image is of a scent released at the goal spreading outwards until the robot picks it up and follows it to its source. This involves no shared representation between the service and its user. Another advantage of the use of this sensor rather than returning an explicit optimal path is that if the robot wanders off course, the path does not have to be recalculated.

9. Location seeking example

It is convenient to control the robot's motion by feeding a target to a "location seeking" module, and giving that module responsibility for doing everything necessary to get as close as possible to the goal- navigating around obstacles, backtracking out of dead-ends, etc. This section describes one strategy such a module can use, assuming the existence of the cartographic system described in this paper.

Initially, the robot tries to move directly towards its target. If it strikes a boundary, it will start following that boundary in whichever direction seems "best"- whichever direction seems to require the least deviation from the robot's current path, at least in that locality (since that is the only area it can sense or evaluate from the map). It will continue to follow the boundary until conditions become suitable for it to resume its path towards the target. This will occur if the boundary turns sufficiently to no longer be an obstruction. With this simple strategy, the robot is able to negotiate many obstacles. However, it may lead to cyclic behaviour for obstacles with certain shapes, such as the one in Figure 9-1.

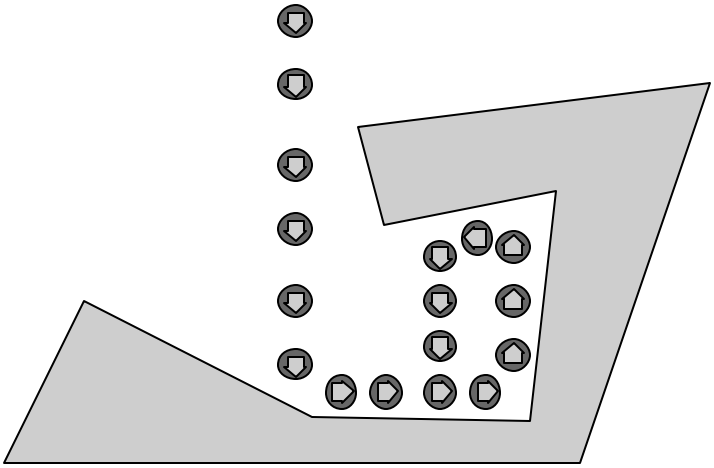

Figure 9-1: Obstacle makes robot loop back on its path

A loop can occur if the robot meets an obstacle, follows its boundary until it again becomes possible to move in the target direction, and then runs into the same obstacle, hence entering a cycle. One simple solution to this, using the "familiarity" virtual sensor generated by the cartographic system (see Section 8.1, page 9), is to make the robot alternate in the direction it chooses to move in after it hits a boundary at a familiar location[7].

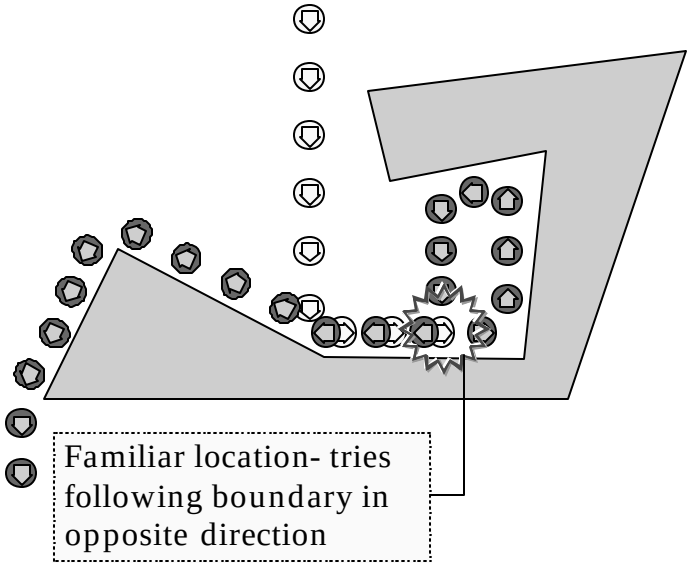

Figure 9-2: Robot uses familiarity to avoid cyclic behaviour

This simple improvement is enough to allow the robot to navigate most common boundaries. Looping can still occur, however, in situations like the one shown in Figure 9-3, where the obstacle resembles a "cave".

[7] That is, any location whose familiarity indicates that it was visited after the robot started seeking its current target.

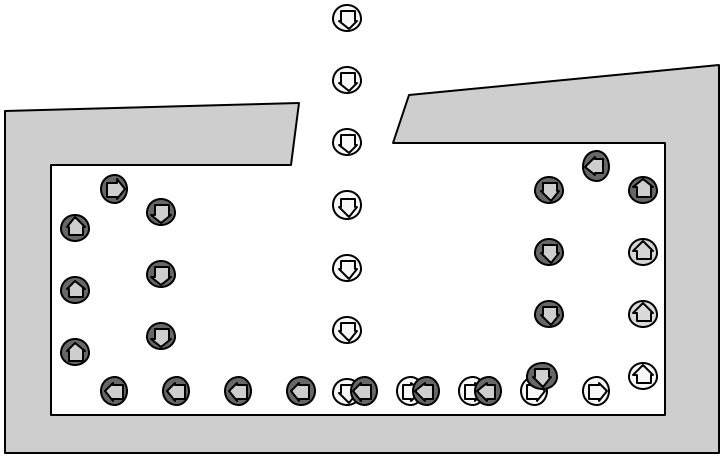

Figure 9-3: Behaviour cycle in "cave"-shaped obstacle

Here, because the boundary-following strategy turns the robot back on its path twice, it again enters a loop. Cases like this can be handled using the familiarity sensor in a more general way than described above. The robot should take an opportunity to return to moving in the direction of its target only if it is following a boundary in *unfamiliar* territory. Then, if the robot is following a boundary in *familiar* territory- territory that it has moved through before while seeking the current target- it "knows" that it should not return to moving in the direction of its target even if it seems desirable to, since this is what it would have done last time it was there. Instead it should wait until it reaches unfamiliar territory again, and then turn whenever appropriate. This results in an expanding search that can get the robot out of awkward situations like the "cave" obstacle, as shown in Figure 9-4.

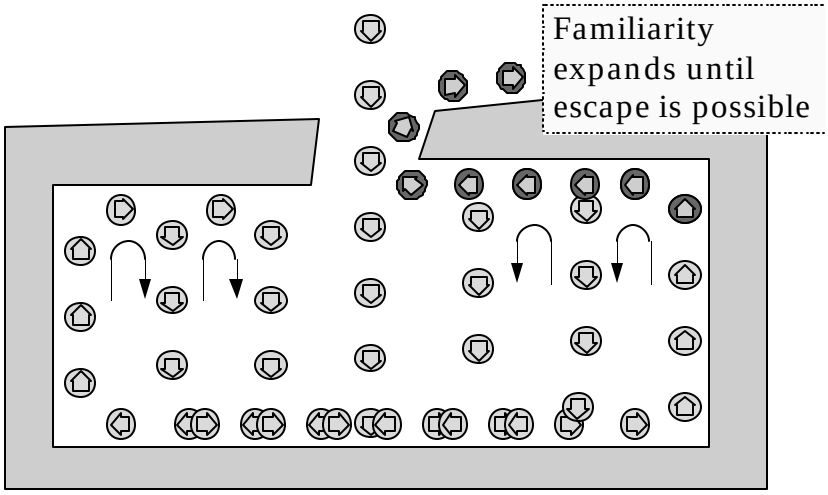

Figure 9-4: Robot escapes from cave-shaped obstacle

It is easy to see that this approach should never result in looping behaviour. If the robot makes a decision to turn at a particular location, and that decision results in it looping back to that same location, the next time around that area will be familiar to it so it will not turn there again.

The robot can "hedge its bets" by using the Goal Seeking service of the cartographic system in the background to search for a route to the target (see Section 8.2, page 10). This searching process is slow, but if the robot is delayed backing out of dead-ends as it tries to get to the target the search will have time to succeed, and the robot can then switch to simply following the "scent" virtual sensor to the goal. It can

always revert back to using location seeking if the path found by goal seeking proves to be inaccurate due to changes in the environment.

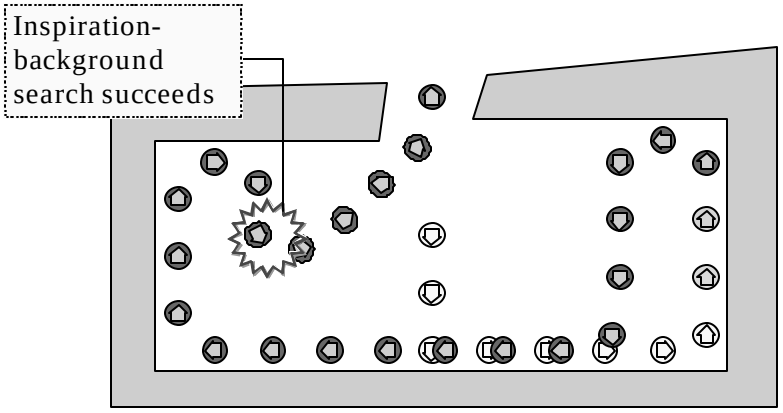

Figure 9-5: Use of background search

Goal seeking is only suitable for use as a safeguard, not as the robot's main strategy for finding a route to the target. Lower-end robots have insufficient processing power to implement such a search in anything approaching real-time, so they would have to "sit and think" before moving at all[8]. Also, since planning based on a map constrains the robot to moving through regions it has already explored and mapped, opportunities may be missed that the "physical search" approach could have taken advantage of.

10. Sample results

This section presents some results from the use of the robot implementing the cartographic system described in this chapter.

10.1 Use of landmarks

10.1.1 Ablation study

In this test case, the utility of the use of landmarks is demonstrated by removing them.

[8] For the robot this work was implemented on, a feature was added so that the search could be speeded up a hundredfold by an external request from the user, with significant degradation of the real-time performance of the robot (it grinds to a halt for a few moments).

Test case	Standard maze, edge following
Feature demonstrated	Robot performance without use of landmarks

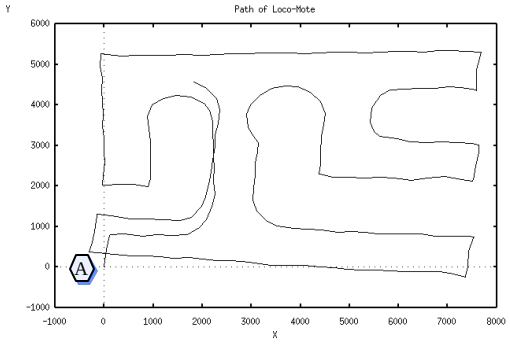

Figure 10-1: Position drift

In the diagram opposite a trace of the robot's best-guess of its position over time is shown while edge-following a closed boundary. Note that when it reaches its starting point around A, error has accumulated in its position sense and it is starting to deviate significantly from the robot's true position.

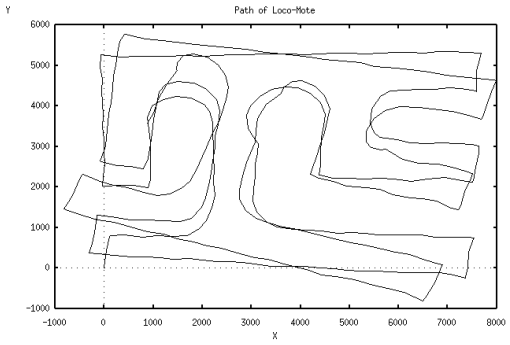

Figure 10-2: Aliasing of topological features

The error in the robot's best-guess at its position continues to accumulate. Here the robot is circling the boundary for the third time. Topological features have started to overlap, and the robot's map is useless-worse than useless, misleading.

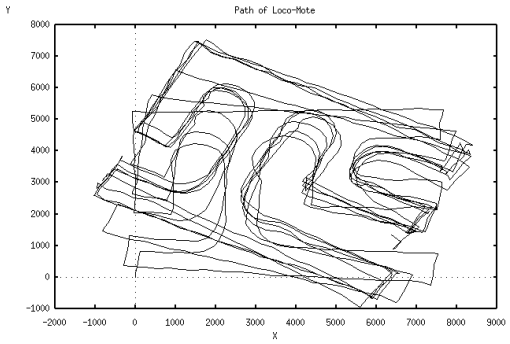

Figure 10-3: Recovery when landmarks reintroduced

The robot was left to continue circling for a while, and then the use of landmarks was turned back on. The trace shows that the robot eventually started to develop a consistent map of the boundary it was tracing. This is seen where a cluster of lines make a seemingly thicker line, showing that the robot started to trace the boundary in a consistent manner.

11. Summary

In this chapter, a complete cartographic system has been developed that is capable of constructing and maintaining a map of the robot's environment in real-time, and with the use of short-range proximity sensors only. The representation scheme used was based on units called "markers". The map consisted of a collection of these markers- records of the robot's experiences at particular locations- rather than being an explicit model of its environment. Strategies for maintaining this collection of markers in a state that reflected the condition of the environment were discussed in detail. Then the issue of landmark recognition was addressed, showing that the robot could maintain an estimate of its position that remained consistent relative to its environment over time. This is the ultimate test of a cartographic system. Examples of services the map could provide were presented that allowed the map to be used without requiring any knowledge of how it is represented. An example of robot behaviour using the map to escape from behaviour cycles, back out of dead-ends, plan routes to targets, was shown. Finally, results were given.

12. References

[1] Kortenkamp, D., Weymouth, T.: Topological mapping for mobile robots using a combination of sonar and vision sensing, Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), July, 1994.

[2] Mataric M.J.: Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior, JETAI Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents, Vol. 9, Nos. 2-3, Hexmoor, Horswill, spec.iss. on Software Architectures for Physical Agents, Vol. 9, Nos. 2-3, 1997.

[3] Sahota M.K.: Reactive Deliberation: An Architecture for Real-Time Intelligent Control in Dynamic Environments, in Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI Press/MIT Press, Cambridge, MA, pp.1303-1311, 1994.

[4] Mondada, F., Franzi, E., and Ienne, P. 1993. Mobile robot miniaturisation: a tool for investigation in control algorithms. Proceedings of the 3rd International Symposium on experimental robotics, Kyoto, Japan, October 28-30 1993, Springer Verlag, London, 1994: 501-513

[5] Arkin R.C.: Integrating Behavioural, Perceptual, and World Knowledge in Reactive Navigation, Designing Autonomous Agents, pp. 105-122, The MIT Press: Cambridge, MA, 1990.

[6] Payton, D. W.: Internalized plans : a representation for action resources, Journal of robotics and autonomous systems(1&2), June 1990, Vol. 6, pp. 89-104, 1990.

[7] Brooks R.A., Stein L.A.: Building Brains for Bodies, Autonomous Robots, 1, pp. 7-25, 1994.

[8] Gat, E.: Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots, Proceedings of the 10th National Conference on Artificial Intelligence, pp. 809-815, MIT Press, July 1992.

[9] Moorman, K., Ram, A.: A Case-Based Approach to Reactive Control for Autonomous Robots, AAAI Fall Symposium on AI for Real-World Autonomous Robots, Cambridge, MA, October 1992

[10] Maes P.: Situated Agents Can Have Goals, Designing Autonomous Agents, pp. 49-70, The MIT Press: Cambridge, MA, 1990.

[11] Agre P.E., Chapman D.: What are plans for?, Maes P. (ed.) New Architectures for Autonomous Agents: Task-level Decomposition and Emergent Functionality. MIT Press, Cambridge, Massachusetts, 1990.

[12] Ferrell, C. 1993. Robust agent control of an autonomous robot with many sensors and actuators. MIT AI Laboratory Technical Report 1443

[13] Ram, A., Santamaría, J. C. Continuous Case-Based Reasoning, AAAI Workshop on Case-Based Reasoning, Washington DC, July 1993. 1993.

[14] Kaelbing L.P., Rosenschein S.J.: Action and Planning in Embedded Agents, Robotics and Autonomous Systems, Vol. 6, No. 1; also in Designing Autonomous Agents, The MIT Press, 1991

[15] Anderson, T. L., Donath M. Animal Behavior as a Paradigm for Developing Robot Autonomy, Designing Autonomous Agents, pp. 145-168, The MIT Press: Cambridge, MA, 1990.

[16] Beer, R. D., Chiel, H. J., Sterling, L. S. A Biological Perspective on Autonomous Agent Design, Designing Autonomous Agents, pp. 169-185, The MIT Press: Cambridge, MA, 1990.