EXPLORING AND MAPPING CONFINED AND SENSOR-CHALLENGED ENVIRONMENTS WITH PROPRIOCEPTION OF AN ARTICULATED MOBILE ROBOT

by

Jacob Everist

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

March 2014

Dedication

to family

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this dissertation, we develop an algorithmic approach for robotically mapping confined environments using internal sensors only. Some challenging environments such as underground caves and pipes are impossible to explore and map with the currently external sensor technology. Such environments can be too confined or too hostile for humans to explore. Robotic solutions are possible, but such environments often cause available external sensors to fail. A robotic solution that can explore and map without external sensors would allow access to previously inaccessible environments.

For example, an approach that is invariant to ambient fluid (air, water, muddy water, etc.) would significantly impact cave exploration and underground science by the access to still deeper and smaller cave arteries regardless of whether they are submerged in water. Search and rescue operations would have additional tools for locating survivors in complicated debris piles or collapsed mines. Utility service workers would be able to map and inspect pipe networks without having to excavate or interrupt services.

1.1 Limitations of Exteroceptive Sensors

Exteroceptive sensors are sensors that are affixed externally and designed to sense information about the robot's environment. They're biological equivalent include such things as vision, hearing, touch, taste and smell. The robot equivalent include touch, cameras, sonar, and laser range-finders.

Currently we cannot effectively explore and map these environments with exteroceptive sensors. Current state-of-the-art mapping methods depend on exteroceptive sensors such as vision, range, or touch sensors that are sensitive the ambient properties of the environment. Sonar and laser-range finders are sensitive to the ambient fluid properties for which they are calibrated. They are also often ineffective due to occlusions, distortions, reflections, and time-of-flight issues. Changing visibility, lighting conditions and ambient fluid opacity impact the usefulness of vision sensors. Hazardous environments may also damage fragile external touch sensors.

1.2 Tracking Position

In addition to the challenges of sensing the environment, we also have the challenge of tracking the robot's position in the confined environment. Due to the enclosed nature of the environment, global positioning information is often unavailable. Therefore, any mapping method will need an accurate motion estimation approach for the robot. Wheeled dead-reckoning may be very difficult or impossible if the environment is highly unstructured or otherwise untenable to a wheeled locomotion approach. Even a well designed motion estimation approach will be susceptible to accumulation of errors and will require localization techniques that will reduce and bound the positional error of the robot.

1.3 Proprioceptive Sensors

Given these restrictions, developing an approach that will work on all the most challenging conditions forces us to rely on a strictly proprioceptive approach for collecting environmental information and building the map. Such proprioceptive sensors include accelerometers, gyros, INS (inertial navigation systems), and joint angle sensors. Examples of biological proprioception includes sense of hunger, fatigue, shortness of breath, pain, sense of hot, sense of cold, perceived configuration of the body, and numerous others.

Exteroceptive mapping approaches directly sense large sweeps of the environment at a distance and allow us to make multiple observations of environmental features from multiple poses, integrating this into a map. With a proprioceptive approach, the information is by definition internal to the robot's body and local to the robot's position. Any inferred information about the environment is limited to the immediate vicinity of the robot's pose. The challenge is to explore the environment and construct a map with fundamentally less information.

1.4 Void Space and Features

Limited information about the immediate vicinity of an articulated robot can be obtained by sweeping the robot's body to cover all of the void space that is reachable. By taking posture snapshots while the robot is sweeping and plotting them into an image, we can create a rudimentary void space image of the local environment. The challenge then becomes how to utilize this particular type of data in a robotic mapping approach.

In the exteroceptive approach (external sensors), one would use the ability to sense landmark features at a distance and identify and merge their position across several views. However, in a proprioceptive approach, all sensed information is local. Not only will the opportunities for spotting landmark features between consecutive views be limited, but we must also define what exactly we will use as landmark features. In exteroceptive mapping, the boundaries of the environment are sensed and landmark features are extracted that represent corners, curves, structures, and visual characteristics. With proprioceptive sensors, we present the different kinds of features that can be extracted and used from void space information.

In the event of visiting the same place twice, we wish to detect the sameness of the location and correct the map to make it consistent. This is often called the loop-closing or data association problem. In the case of exteroceptive mapping, this is often achieved by finding a correspondence between sets of landmark features that are similar and then merging the two places in the map. In the proprioceptive case, the availability of landmark features is scarce. The current best-practice methods for solving the data association problem depend on an abundance of landmark features with which multiple combinations of associations are attempted until an optimal correspondence is found. In our approach, with only a few landmarks to use, performing data association in this way becomes ineffective at best, destructive at worst. We determine a different approach to data association using void space information.

1.5 Related Work

The mapping of confined spaces is a recurring theme in the research literature. Work has been done to navigate and map underground abandoned mines [Thrun04] with large wheeled robots traveling through the constructed corridors. Although the environment is large enough and clear enough to make use of vision and range sensors, its location underground makes the use of GPS technologies problematic. This work can be applied to confined environments where exteroceptive sensors are still applicable.

Robotic pipe inspection has been studied for a long time [Fukuda89], but usually requires that the pipe be evacuated. The robotic solutions are often tailored for a specific pipe size and pipe structure with accompanying sensors that are selected for the task. Often the solutions are manually remote-controlled with a light and camera for navigation.

Other work has focused on the exploration of flooded subterranean environments such as sinkholes [Gary08] and beneath sea-ice or glaciers. This involves the use of a submersible robot and underwater sonar for sensing.

Other work has approach the problem of identifying features in the absence of external sensors. This work comes from the literature on robotic grasping and is described as contact detection. That is, the identification and reconstruction of object contacts from the joint sensors of the grasping end effector. Various methods to this approach are described in [Kaneko], [Grupen], [Haidacher], [Mimura].

Our earlier paper [Everist09] established the basic concept of mapping using void space information with a rudimentary motion estimation technique. This early work attempted to merge the information from contact detection with the new void space concept. We did not yet have an approach for localization, feature extraction, and data association.

Another researcher [Mazzini11] tackled the problem of mapping a confined and opaque environment of a pipe junction in an oil well bore hole deep underground where high temperature, high pressure, and mud make external sensors infeasible. Mazzini's approach focuses on mapping one local area of the underground oil well. They use physical probing with a robotic finger to reconstruct the walls of the pipe. The robot remains anchored and immobile so there are no need for motion estimation. Our dissertation differs from this work in that we use void space information instead of contact or obstacle information, our robot is mobile instead of anchored to one location, and that we are exploring and constructing a complete map instead of one local environment of interest.

1.6 Approach

To address these challenges, our approach is to develop algorithmic methods for utilizing proprioceptive joint angle information on an articulated mobile robot to enable exploration and mapping of a confined environment. From a series of complete posture information sets, we build up void space information into a posture image and integrate it into a whole map. We hypothesize that with the posture images taken at multiple robot poses in the environment, we can integrate them into a map; we believe this approach will serve as effective alternative for mapping confined environments when exteroceptive sensors have failed or are unavailable.

Our claim here is that the type of robot used and its method of locomotion are not important so long as the robot is articulated and with many joints. The more joint sensors the robot has, the better capable the robot is of mapping.

In this design we use a simulator that enables us to rapidly develop and test different algorithmic techniques without becoming overly worried about the hardware and locomotive aspects of the robot. We use a simplified flat pipe-like environment with a snake robot that is capable of anchoring to the sides of the walls and pushing and pulling itself through the pipe on a near frictionless floor.

We do not simulate any sensor-inhibiting fluid in the environment or sensor-failure scenarios, nor do use any type of external sensor, be it touch sensors, vision sensors, or range sensors such laser range-finders and sonar. Additionally, no global positioning system is used since it is not practical to expect GPS to function underground. The only sensors we permit ourselves to use are joint sensors.

For capturing the posture images, we examine a number of difficulties and pitfalls to capturing this information. Notably, the prevalence of slip of the robot can corrupt the posture image because the robot loses its stable frame of reference to the environment. We develop a number of techniques for safely capturing the void space information, reducing the likelihood of slip, detecting slip, and mitigating the damage if slip should occur.

We examine several possibilities for extracting landmark features from the posture images. We try several traditional forms of features such as corners and edges, but in the end we develop our own macro-feature which is the overall shape of the local environment captured by the posture image and reduced to the form of a medial axis.

When the robot is moving through the environment, we must also have techniques for estimating the motion between the poses at which we capture the posture images. We show a highly accurate but very slow method [Sec Motion 1] for our particular snake robot morphology, and compare it to a coarse but fast motion estimation method [Sec Motion 2]. We discover that the coarse method is sufficient so long as we rely on the macro-features of the posture images to constrain the possibilities.

Given these basics, we then delve into the heart of the matter: the actual integration of poses, motion estimates, and posture images into a coherent map. We develop and experiment with a few techniques using the constraints from the macro-features in a variety of ways [Sec Constrain 1]. We finally settle on an aggregate pose constraint method where the whole set of past poses is used to generate a feature that constrains the next pose [Sec Constrain 2].

We show how our mapping approach works in a variety of different environments for example, a single path pipe where there are no branches in the environment. The environment only contains a single path that includes a different variety of turns and angles. We then complicate matters by adding in different types of junctions such as T-junctions, Y-junctions, and X-junctions. We do not yet consider environments that have loops in them.

The adding of junctions adds significant complications and requires us to have a better understanding of the structure of our map. We establish a new type of mapping structure called a shoot map. A shoot is a section of the environment that is a single path. The shoot has an origin and a termination point which indicates either the boundary of the environment or more space yet to be explored. New shoots are created when a junction is detected, where the new shoot's origin is at the location of the parent shoot where the junction was detected. That representation allows us to deal with the restrictions on the type of information we can acquire and the often incompleteness of the posture image capture with respect to the actual local environment.

Finally, given the fundamental mapping structures and methods we have developed, we cast them into a probabilistic framework. This allows us to keep track of different map possibilities, discarding improbable ones, and selecting the most probable. Our mapping approach has certain ambiguities in the type of environmental information we can acquire. The probabilistic framework keeps track of these ambiguities until they can be resolved when more information is acquired.

Chapter 2

Sensing Space

2.1 Proprioceptive Sensing

- Sweep the space. Limited using the geometry of the robot.
- Posture Images I_1:t
- More here

Integrate (Posture vector \phi_t, t) = posture image

Breaking up the posture image into forward sweep and backward sweep. The transition between backward anchoring and forward anchoring causes the reference frame to slip and the posture image to be distorted. We show that is better to have two separate images that do not disrupt the anchored reference frames.

In the event, we have local correction that sanitizes the data. If there should be a sudden breach of the arm outside of the containing polygon of the posture image, we classify this as a slip and adjust the local coordinate system to keep the instantaneous posture within the boundaries of the polygon.

Should the data be too hopelessly corrupted from a bow tie effect in the posture image, we can reject the overall data and take the instantaneous shot of the current posture as the only data. This gives a sparse snapshot of the environment and leaves the exploratory data out, but if will capture the gross shape of the local environments.

2.2 Pose Map

Actions and posture images are raw input.

Images I_1:t

Actions: U_1:t

Find poses P_1:t to solve mapping problem

Combine the elements to get a naive map. Simple motion estimation and plotting of poses and posture images to a grid map. No capability for correction.

Chapter 3

Control and Locomotion

3.1 Control Methodology

Backbone curve fitting, IK method, segmented behaviors

3.2 Behaviors

Locomotion steps, extensions, path-following, anchoring

Chapter 4

Defining Position

4.1 Defining Pose and Direction

Articulated robot has no high-inertia center of mass to serve as its coordinate frame. Require method to establish forward direction of an articulated robot.

Establishes stability of robot frame under error of anchoring.

4.2 Motion Models

Naive motion model (fast time, low accuracy) Self-Posture motion model (cost of time, increase accuracy) *Step model with inter-pose overlap invariance (fast time, medium accuracy)

Examples of motion estimation results.

Naive + GPAC Frame Self-Posture + Body Frame *Pose Step + GPAC Frame (discussed later)* Shoot Step (discussed later)

Chapter 5

Environmental Landmarks

5.1 Finding Landmarks

Seeking landmark features for which we can correct the map and localize the robot. Most approaches use point features such as corners, environmental landmarks such as doors, visual features. Our options are very limited because we do not have sensing at a distance capability. All things that we can sense must be within touching distance. Any feature we choose is highly local.

One option is to find corner features by probing the walls. There are two approaches: 1) deliberate contact probing of the wall to produce corner features, 2) serendipitous corner features. Problem: 1 is feasible but slow, see [Mazzini]. 2) is also possible but the false positive rate is too high. Pull some data out of my archives.

Also try and find obstacles and openings . That is, non-obstacle openings. This presupposes that we can distinguish obstacles. Only contact probing methods can truly classify an obstacle boundary. Again, these methods suffer from time-consuming actions. Furthermore, this does not exactly give us a true positive for open space. So door openings cannot be a landmark since we cannot detect them directly.

What about the walls of the environment? The shape of the obstacles? This requires that we fully mark the boundaries of walls and obstacles. Either this is done through laborious contact detection or we make assumptions about what are obstacles based on the data we receive serendipitously. Given a posture image, we could make varying degrees of assumptions about where the boundaries of the environment lie. None of them are likely to be correct, but we can use this information as a feature. However, this approach suffers when our probing actions are deficient for a particular pose. We receive a degenerate posture image that leaves out a lot of data.

Another approach we could take is a sort of approximation and averaging of boundaries of the posture images. This can be accomplished by the use of the alpha shape algorithm developed in [3]. It is a form of generalized convex hull algorithm that is capable of non convex polygons. This can give us our desired approximation effect for identifying the location of walls.

5.2 Corner Examples and Results

Corners are high false positive and too many false negatives.

Requires tuning of parameters, but does not guarantee correction selection.

Corners through serendipitous detection are not seen very often. Will not be seen through consecutive poses or even being in the same location. Not guaranteed to view.

Information too sparse to be useful. False correlation is very costly.

5.3 Wall Detection

Contact detection literature. Reference 2009 paper.

Time spent probing walls. Amount of information is broken up. Example information should walls from contact detection.

Punch probing. Hit the wall and measure the error incurred by the joints in the control loop. Avoids the need for torque sensor. Uses existing control algorithm parameter as error terms.

Broken information, time-consuming. Error prone and often will mark free space as wall. Not detailed enough for a landmark feature.

5.4 Pipe Shape, Macro Features

Use the contours and shape of the occupied space as our landmark feature.

Introduces strong lateral correlation, but weak forward-backward correlation. If the local environment is curved in any way, this provides stronger localization.

1st order curve is straight line. Only coaxial localization between poses 2nd order curve is a constant curve. Localization to any place with same curve constant. *3rd order curve is changing curvature. Localization to unique point with particular change profile.s

5.5 Macro Feature Extraction

1. curve of anchor points, represents walls. Does not include sensed data

Posture Image -> Alpha Shape -> Medial Axis -> Characteristic Curve

Medial axis is generated from an image and a tree is built. I most cases the tree has only two leafs and is a path graph, but sometimes the tree has more than two leafs and represents more complex space.

Represents local description of space.

Like scan-matching, we can use curves and ICP to find an alignment of the poses. The curves themselves becomes the landmarks.

5.6 Different Macro Features

The possibilities for curves representing the overall macro structure of the local environment are as follows:

1. Curves fitting through the anchor points of the articulated robot. Anchor points are the confirmed contacts and can approximate the walls. However, these are sparse data points, and cannot handle complex environments such as junctions or

gaps between the anchor points. Data is sparse and not well-representative of the environment.

2. Polygon created from alpha shape of posture image. Use the edges of the polygon and perform scan-matching of edges to edges. Problem, how to select the points pairs between two polygons. Use angle filter and match two sides together. Is sensitive to missing data. Unsweeped gap will distort the polygon. Performing matches on a distorted polygon will result in poor ICP fitting.

3. Given alpha shape, compute the medial axis of the interior. This is known to be a reduction and representation of the the space. This approach is resistant to noisy edges. Shown are some example of the results with simple single path pipes, and pipes with some junction features. Notice that the information on the width of the environment is lost using this approach.

Chapter 6

Pose-Based Mapping

6.1 Inter-Pose Correction

FIXME: apostrophe

Now that we have curve representations for each pose's posture image, we can apply an overlapping constraint. That is, assuming the invariant condition that two poses' posture images partially overlap each other, in what way(s) do the medial axis representations overlap each other. The best way to do this is with a generalized ICP algorithm, with an initial guess that is generated from the motion estimation method.

Although the two curves are guaranteed to overlap in some contiguous way, this does not mean they all overlap in the same way. The medial axis could be more than just a single curve segment, but a graph if their are some junction-like elements. Then the question is, how do you select which parts of the medial axis to match to the consecutive pose medial axis.

In a multi-curve medial axis, we can try all possible combinations. The result whose offset is the most consistent with the initial guess should be the proper result. In practice, a really great test for measuring the correctness of a particular ICP fit is to select the one with the lowest angle difference from the initial guess. This assumes that initial guess of the angle was reasonable. Getting good angle guesses is possible with the GPAC frame in Defining Pose and Direction section.

Technically, the relative offset between two poses is a 3 dimension problem (x,y,\theta). However, this can be reduced to a 2 dimensional problem by defining an intersection point between the two curve segments. If the point on curve A remains fixed, if we vary the point on curve B and the angle of curve B, then the ICP problem is in a 2 dimensional space. This intersection constraint enforces a particular type of solution that ensures the curve segments are overlapped.

So long as the initial guess from the motion estimation method is reasonable, performing an ICP fit of the two posture medial axes enforces a sanity check on their invariant overlapping.

Now that we have two medial axes, we show that the step motion model is defined by a constant arc length displacement between the GPAC origins of two consecutive poses. The step motion model combines the displacement estimate with the constraints of the locally perceived environment. Of course, the direction of that estimate may be erroneous if their are multiple splices of a multi-medial axis tree. Particularly, if we are in a junction that is sensed by a pose with two curves going to left or right, a displacement can choose to go left or right but the correct result is only determined after comparing macro-feature fit of the subsequent pose.

The step motion model is a combination of local tracking of macro-features of the environment and a model of the robot's average displacement during locomotion. As shown in the figure, the results of the naive motion model, the self-posture motion model, and the step motion model show clearly the best answer for initial guess of the robot's pose between consecutive poses. See our earlier discussion in Motion Models

6.2 Paired Pose Correction

For two poses that are theoretically at the same location but are split for the forward and backward sweep, we need some way for reconciling their relative offset. Taking the raw pose data and posture images, we see there are some discrepancies caused by the slip during the anchor transition. We can fix this by performing an overlap fit using the same method between consecutive steps in inter-pose correction.

The initial condition will be the poses located on top of each other and their medial axes intersecting near the origin points. After performing an ICP fit to change the angle and the arc length displacement in the neighborhood, this creates a reasonable estimate transform between the two.

6.3 Constraint Mapping

Combining the motion model estimates and the inter-pose constraints, we can create an improved map that is better than just the simple motion estimation maps. The inter-pose constraints give a better trajectory for the environment. However, they are still deficient in a number of ways.

They do not provide any means of resolving the unique correlations of a junction, nor do they provide any consistency when backtracking through a local environment we have been before. That is, they do not have provide a solution for the loop-closing or data association problem.

A possible solution to this is to attempt to make additional hypothetical constraints between poses that are in the same neighborhood. Following [Olson], we can establish covariances between the constraints and generate consistency matrices to determine the most optimal constraint hypothesis set. This is the Graph SLAM [Probabilistic Robotics] approach that use a pose graph representation for generating the map.

The weakness of our problem domain is that we do not have available landmark features that fit into this framework. Landmark features are given their own positions and then observed from multiple robot poses which are then correlated. 1) Our data does not have point features with positions, we have macroscopic shapes of the environment. 2) Secondly, we are not guaranteed to observe the feature between consecutive poses or even poses that are in the same location.

The Graph SLAM approach becomes increasingly difficult because there is no clear way how to generate covariance matrices between poses. We could generate the covariance with a coaxial variance for greater variance in arc distance and low variance for lateral offset off the overlapping axes. This is clear for simple straight posture images.

However it is less clear for curved or multi-axis junction environments. Furthermore, using a graph optimizer can result in two hard constrained poses to be corrected to be off-center their medial axes to globally minimize the overall error. This is not the type of corrections we want to see since the local constraint is so certain in only one dimension. Any correction method should take advantage of that.

6.4 Aggregate Pose Constraint Mapping

Instead of creating geometric constraints between individual poses and struggling to search for which acceptable matches between poses, another approach we can take is to make a constraint between a new posture image and all of the other posture images in aggregate. In this way, we can avoid the difficulty of making individual pairwise constraints between poses.

The approach we take is to find the union of all past posture images together and compute the alpha shape from that. From the alpha shape we then compute the medial axis. This results in a much larger version of the medial axis curve representation.

To constrain the pose of the new posture image, simply perform ICP between the pose's medial axis curve and the past whole map medial axis. The result can be seen in the figure X. Like the inter-pose constraint, we have the invariant condition that the new pose's medial axis curve partially overlaps the global medial axis. Therefore, there exists a section of the global medial axis curve that the new pose at least partially overlaps. For poses that are backtracking through previously visited territory, the new pose will be completely overlapped by the global medial axis.

This is a maximum likelihood mapping approach. That is, the mapping method looks for the best possible result for the current pose. All past poses are fixed and unable to be changed. Should we ever make a significant mistake in the placement of the current pose, this will be permanently reflected in the map for the future.

Even using this simplistic approach, a significant translational error along the axis of travel in a straight section of the environment does not have severe consequences to the map. Along a straight section of the pipe, the current pose can be placed anywhere along that section with little consequence to the structural and topological properties of the map. The only thing that is affected is the current best estimate of the robot's pose.

The mapping system is able to localize the pose much better with respect to curved shape features in the environment. If there are sharp junctions or curved sections, the current pose can be localized with much better accuracy.

Once the current pose's posture image given it's most likely location, it is added to the existing set S and the union->alpha shape->global medial axis is recomputed. If the robot is pushing the frontier of the map, this will result in an extension of the medial axis. If the posture image pose is an incorrect position, the resultant global medial axis can be corrupted and result in kinks (distortions, discrepancies). These kinks can compound future errors and result in even further distorted maps.

To compute the raw medial axis of a path, we must take the union of the set of its constituent poses. For every pose, we take the set of points that constitute the polygon representing the pose's alpha shape in the local coordinate frame. For every point set B of every pose P, we transform the set B to B' in the global frame. The union of all sets B' is the resultant point set C.

We then compute the alpha shape of C with a radius of [0.2]. The resultant alpha shape is mapped onto an image of appropriate dimensions with a pixel size of [0.05]. All of the pixels contained within the grid framed alpha shape are set to 1 and all others are set to zero. The resultant image is input to the computeMedialAxis function.

The medial axis result will look similar to Figure [X] with overlap into the parent path and possibly multiple branches because the robot overlaps the junction in different ways for each pose. We cannot use the medial axis in this form because it does not logically correspond to the data representation we are looking for. Union path tree of all the paths overlap each other in multiple ways. Our goal is to extract the path fragment that extrudes from its parent path at the designated branching point. Using only this section, this is our representation of the child path.

Identifying the junction requires some finesse. Given the divergence point on the branching pose, find the closest point on the union path tree to the local divergence point. This becomes our theoretical branching point. We create a branching leaf along the desired direction if the union path tree is too curvy or does not represent the sharp junction structure that we desire to represent. The comparison of the theoretical leaf and the generated path is shown in Figure [X].

We then call the trimPath() function which will prune the tree at the branching point, keep the section of the path tree that overlaps the our desired section of environment, and ensures that the result is a single path and not a tree, pruning off any extraneous leaves.

The first thing we do is find the longest path between a pair of leaf terminals on the path tree such that the longest path passes through the branch point at the target angle.

The topology of a path is computed from the union of all of the poses' alpha hulls classified into that particular path. The hull is populated with points in a grid and the medial axis is computed using the skeletonization (?) algorithm.

Given the set of points that are the skeletonization image, the pixels are converted into nodes of a graph, and edges are added between them if they are adjacent or diagonal neighbors. The pixel indices are converted back into global coordinates to give us a set of points that are nodes in a graph, edges connecting to their neighbors.

We reduce the graph first by finding its MST. Then we prune the tree by snipping off one node off of leaves. This removes any obvious MST noise artifacts but preserves real branches of the tree.

Paths are computed between each pair of leaves. We classify each node that has a degree > 2 as a junction and possible branch point for the target path we are trying to build. On each leaf-pairwise paths, we find the point on that path that passes through the junction.

In the case that there exists no junction or branching point close to the pre-existing branch point, we create a separate graph that includes a theoretical branch with the previously computed parameters. These parameters come from past computations of the path topologies. We know that a branch exists but it may have been erroneously smoothed out due to the addition of new poses in non-optimal locations.

The theoretical branch is computed by taking the orientation of the historic junction point, finding the closest node on the graph to the historical junction point, and extruding a new branch on the tree from this point. A series of uniform points are computed to represent this branch that terminates at the boundary of the alpha hull.

Each leaf-to-leaf path on the medial axis is extrapolated at the tips to intersect and terminate at the boundary of the alpha hull. This adds further usable information to the produced curve and in practice is a correct extrapolation of the environmental topology.

We select the longest leaf-to-leaf path that includes the branching arm of the junction from its parent path. The portion of the path that branches is the portion that we are interested for the current path. The portion that is not branching is overlapping its parent path. This information can be used to help localize the branching point with respect to the parent path.

We then compute the trimmed version of the computed topology path. At the junction point, the path is cut and only the remaining portion that forms the branching arms is kept. This portion is the final path portion and does not include any portion of its parent.

When using these path fragments, we splice them together in various hypothesized configurations to determine if our robot's posture fits into any previous paths and junctions. This splicing is helpful for localization and navigation when exploring the environment.

Chapter 7

Mapping with Junctions

7.1 Map Algorithm

1) Estimation motion of step

2) Collect posture image

3) Determine if there is a branch, if so, fork process (yes/no cases) (NEW)

4) Add pose to best fit shoot, generate new map (NEW)

5) Localize pose to most likely location in existing shoot map, fix map (NEW)

6) go to 1)

ESTIMATE MOTION - estimate position of pose based on motion estimation from the previous fore pose: {f, b}

SENSE ENVIRONMENT - capture environmental information through probing - perform forward sweep for front pose - generate local spline

ESTIMATE MOTION - estimate position of pose based on a zero move estimation from the paired fore pose: {0}

SENSE ENVIRONMENT - capture environmental information through probing - perform backward sweep for back pose - generate local spline

CLASSIFY POSES TO PATHS - Find each best fitting path splice for the front and back pose's local spline - If the pose's local spline does not diverge from the path splice, we add the pose to the splice's constituent leaf path(s) - Otherwise, if it diverges and branching parameters are sufficiently unique, create a new path from the branching point and add the new pose

PATH COMPARISONS FOR SIMILARITY - compare sibling paths to see if they are similar - queue them to be merged if they are similar

- compare parent-child paths to see if they are similar, - queue them to be merged if they are similar

MERGE PATHS THAT ARE SIMILAR - merge sibling pairs - merge parent-child pairs

- recheck for branching events from the constituent poses after merging

LOCALIZE POSE - Localize fore and back poses to their classified paths. - Maximum likelihood localization to classified path

7.2 Junctions

This approach works well for environments that are just a single continuous path. If we encounter an environment with a junction, we will need to modify this approach. One of the challenges of environments with junctions is that they are only partially observable. The robot may be passing through a junction without detecting it as such. Only by passing through all arms of the junction over time are we able to infer the location and properties of the junction.

Given that a junction is not visible on its first pass except for special circumstances, the best approach is to assume there is none and continue to add the pose's single posture image to the global union and compute the resultant global medial axis. Evidence for the existence of a junction is given by a posture image's medial axis curve partially overlapping a section of the global medial axis and then diverging from it at a sharp angle.

We must recognize that there are three possible ways for a new pose's medial axis to fit onto an existing global medial axis. It can either 1) completely overlap, finding a localized position in the map, 2) partially overlap by extension, pushing forward past the termination point of the global medial axis, or 3) partially overlap by divergence, curving off the global medial axis at a sharp angle signifying a junction-like feature.

One of the fundamental challenges of exploring environments with no ability to sense at a distance, there is no quick way to distinguish a wall from the limit of arm extension. That is, when the robot's body sweeps and reaches the limit of its motion, by either collision, slip, or complete extension, there is no way to immediately distinguish that the boundary of its swept space is an obstacle or just more free space. [complete extension is well-defined, therefore, it can be tagged when complete extension is made. Planned motion vs. actual motion.]

Though we can reason and safely map long tubes with hard turns or curved and undulating paths, if we add junctions to the mix such as Y and T junctions, recognizing the encounter and the parameters of the junction becomes especially challenging.

We define a junction to be the confluence of over two pipes at various angles. For instance, an L bend we do not consider a junction, but a T is a junction. To see how this is challenging, imagine the Figure [X] where the robot's body is within the L junction. Similarly, the robot's body is in the T-junction in the same configuration. Both position's give us the shape of the environment correctly. However, we can only partially view the junction at any given time.

In order to sense and completely characterize the T-junction, the robot would be need pass through or back up and make an effort to crawl down the neglected path. If we knew that the unexplored pipe existed, this would be an easy proposition. As it stands, it is very difficult to distinguish an opening from an obstacle.

There are deliberative ways to completely map and characterize the environment as we travel. The work of Mazzini and Dubowsky [cite] demonstrate a tactile probing approach to harsh and opaque environments. Though this option is available to us, this has it's own challenges and drawbacks.

For one, tactile probing is time consuming. A robot would be spending all of its time touching the walls of the environment in a small range and not enough time exploring the environment in the larger range. We decided very early on in our research [Everist2009] that although contact detection approaches have proven their effectiveness in producing detailed environmental maps, they are not a practical solution to exploring complex environments that won't take multiple days to complete.

Similarly, the act of probing and sweeping the environment has the possibility of disrupting the position of the robot and adding defects to the contact data locations. Tactile probing is particularly vulnerable to this since it involves actual pushing against obstacles and possibly disrupting the robot's anchors.

For all of these reasons, it is desirable to find an approach for mapping environments with junctions when the junctions are only partially observable. That is, at best, we do not know if we are in a junction until we come through it a second time. Often times, junction discovery is serendipitous because openings look identical to obstacles in a free space map. Without deliberate tactile probing, junction discovery will always be serendipitous for rapid mapping.

Take for instance the 3-point T junction shown in Figure [X]. There are three possible occupancy states for the robot to fit into the junction. As we can see, states 0 and 1 together give us a complete representation of the junction. However, it is not clear from just the estimated poses and local spline curves of the data that they should be matched up together. There is ambiguity in their relationship since the poses can be arranged in a number of different configurations that represent completely plausible environments.

In order to reduce this ambiguity, we need additional observations of the junctions increase the likelihood that they fit together into a T junction. In particular, if the robot goes from state 0 to state 2 and back to state 1, this would give us a blueprint to fit all of the poses together to create a complete T junction.

Even though the data set we have gives us a very strong case for the detection of a T junction, there are still other possibilities in the event of poor motion estimation. Other plausible environments from the given data are shown in Figure X.

As we can see, the approach of mapping junctions using only free space data has its challenges. There is no full proof way to map a junction using only partial observations with ambiguous resultant data. However, we will see if a probabilistic approach will give us to cut through the ambiguity and focus our efforts on the most likely cases. Furthermore, it could direct our efforts to make a costly targeted contact probing effort at the most likely junction location and resolve all confusion.

When making the hypothesis of a junction, we have some leeway on its location depending on the type of junction states of the composing local medial axes. As seen in Figure [X], if a junction has been hypothesize by the a local medial axis diverging from its parent path at location A with the shown junction state, there are a continum of possible locations it could be at that are completely consistent. Though these possibilities are constrained by the estimated positions, these possibilities give us our search space for fixing the final product.

If we were to finally add in the local medial axis B in the depicted junction state, the freedom of movement gives us the option of resolving any conflicts and making the junction whole with the composed poses. The ambiguity places boundaries on the search space.

We call these unresolved junction arms to be phantom arms. We suspect that a junction arm exists, but we do not know until we actually explore them. Furthermore, we do not know the junction's location until we have an extra pose in the correct junction state, or we have gathered evidence that a junction does not exist and decreased its probability.

7.3 Shoot Map Representation

To represent and keep track of junctions and any instance of branching off from the main global medial axis, we introduce a new map representation called a shoot map. First we define the concept of a shoot to be a single path with an origin and termination point that represents a continuous representation of space with no junctions. In the previous section, our global medial axis of the entire environment represented a single shoot, since there were no instances of branching. Once a new junction is found, a new shoot is created with origin starting at the branching off point from the original shoot and termination point ending at the extent the posture images map the branch's space.

Corresponding to the three cases of how a posture curve overlaps an existing global medial axis, 1) either the pose is contained within an existing shoot, 2) the pose extends the shoot, or 3) the pose is added to a new shoot that branches off. Given the nature of the shoot structure, their is a parent-child relationship between shoots, with the stem being the root shoot. All shoots except for the root, have an origin placed from another shoot.

The curve of a shoot is represented by an ordered series of points that are generated from the medial axis computation of the union of member posture images. We say a pose is a member of a shoot S_p if it has some overlap with S_p but does not overlap a child, S_c, of S_p. That is, if any portion of a pose overlaps a child S_c, the pose is a member of S_c and not S_p. A pose can be a member of more than one shoot if there are more than one child shoots. The member shoots can be siblings or any other combination where the shoots are not direct ancestors or descendants.

The complete representation for a shoot map is the following parameters:

- Poses: X_1:t
- Images: I_1:t
- Shoots: S_1:N
    - Shoot member pose sets, c_1 = {1,2,,k}, , c_N = {k+1,,t}
    - Branch point: b_1:N = (x,y,\theta)

The first half are the same parameters used for the pose map. We add the additional parameters of shoots. A shoot is composed of a set of member poses and a branch point. For a non-root shoot, the branch point is a location on its parent shoot and an accompanied direction.

7.4 Generating Shoots

Compute the union of posture images for a shoot set. Cut and paste each medial axis at the designated branch points between the parent and child shoots.

For each shoot S_j, For each pose k that is member of S_j, Plot posture image I_k to pose X_k

```
Find alpha shape A_j of union of pixels
Find medial axis M_j of alpha shape.
```

For every pair of shoots S_j, S_k, If S_j = isParentOf(S_k) Find closest point on M_k to b_k. Cut section from the extra portion of medial axis starting at b_k in the direction of b_k Add section to trimmed shoot map T_k

```
If isRoot(S_k):
    Add M_k to trimmed shoot map as T_k
```

Medial axis from union of set of posture images of particular shoot S where exists vertices degree > 2. Trim the section closest point to the branch point in the direction oriented by the branch point b_k.

7.5 Adding Poses to Shoots

If first poses, add to root shoot.

Check for branching conditions

Best Overlap, get ordered overlapping path ids

Select child shoot IDs, add pose to the shoot.

MOVE ESTIMATION - estimate position of nodes based on motion estimation from the previous pose

MAP INTEGRATION - integrate the new pose into the map

ADD NEW POSES, CREATE NEW PATHS - Which path or set of paths does this pose currently overlap? getOrderedOverlappingPaths() - of all splices in local neighborhood, find the best fit based on highest contiguity with initial estimated pose - in order of best fits, pick the first that overlaps all constituent paths, that is, does not have paths in the splice beyond the range of the pose sensor data - for the constituent path IDs of the chosen splice, order them according to sequence they are overlapped by the pose curve - return the ordered list of path IDs that this pose curve overlaps

- Does this pose fit on the path splice or does it depart internally? getDeparturePoint() of orderedPathIDs
  - get departure state of the front pathID and back pathID with pose curve
  - if the pose curve does not depart internally, add this node to the leaf path(s)
- If it internally departs, given the departure state of the splice, are the pose departure parameters sufficient for a new branch?
  - is the proposed new branch sufficiently unique
    - is the distance from the parent's branch point sufficiently far? (0.8)
    - is a path sibling's sufficiently close (0.5) branch point not angled in the same direction (PI/6 tolerance)?
    - is the pose curve featured?
    - if all true, then this new branch is sufficiently unique
- If the new branch is sufficiently unique, then create the new branch with the current pose
- Regenerate the paths with the new node data
- select parent and child path pairs
- select sibling path pairs

PATH COMPARISONS FOR SIMILARITY - if sibling pairs, - create a stitched path of parent with section between two child junction points excised and the relative orientations of the junction points are aligned - select common point between parent path and stitched parent path - perform ICP fit between stitched parent and original parent - return the resulting pose offsets for both sibling paths and their relative junction angle - if the angle of both offsets is < 0.5 and the junction angle difference is $< PI/4$, than we queue the sibilngs to be merged

- if parent-child pairs,
  - create a set of splices for the parent path and the child path
    - if grandparent exists,
      - generate gparent/parent splices and gparent/parent/child splices
    - else
      - generate parent splices and parent/child splices
  - select common point between parent splice and child splice
  - perform ICP fit between parent splice and child splice pathOverlapICP()
  - return the resulting offset between the two
  - if the resulting offset angle < 0.5 radians
    - compute the departure state of the two paths given their relative offset getPathDeparture()
    - if one path does not internally depart from the other
      - and if the two paths with their relative offset partially overlap getPathOverlapConditionn()
      - then we queue the child and parent paths for merging

MERGE PATHS THAT ARE SIMILAR - merge sibling pairs - for the pair of sibling paths to be merged - transform the nodes in path1 by the offset from the previous ICP fitting - transform the nodes in path2 by the offset from the previous ICP fitting - perform consistentFit() of each node from path2 into best possible splice (NOTE: should be onto a spliced parent-path1 curve) - for each merged node, perform the "ADD NEW POSES, CREATE NEW PATHS" procedure for checking for branching events and classifying nodes into leaf paths

- merge parent-child pairs
  - select child nodes
  - move them by earlier ICP-fitted offset
  - find the most consistent fit of all set of splices in the node's new neighborhood using consistentFit()
  - change their membership from child path to parent path
- Finally, if the node hasn't been made fitted already to its path, do so now using consistentFit()

FUNCTION consistentFit() - find the most consistent fit in all set of splices in the node's new neighborhood consistentFit() - for each possible splice, find the common point betwen pose curve and splice path - perform 11 guesses of where the pose should be along the length of the splice path curve in the neighborhood of the its starting location - perform ICP between pose curve and path curve for each starting guess - compute the departure state, contiguity, and overlap sum for each final position - discard result if - internal departure exists - terminal departures on both ends of pose curve - contiguity is less than 0.5 - the resultant pose is > 3.0 from original estimated pose - difference between starting guess angle and corrected angle is > 0.7 radians - select the splice and fit location with the best utility given by following equation: - utilVal = (0.5-angDiff2)/0.5 + (contigFrac–0.5)/0.5 + (3.0-dist)/3.0 + 1-isDeparture*0.5 - weighting angle difference from initial guess, contiguity %, distance from starting estimate, and if any internal or terminal departure exists

7.6 Detecting Branch

If a posture medial axis is detected that does not completely fit on the existing shoot, it is possible that we have discovered a branch. It is vital that we properly and detect branching events in order to fit them properly into the map. In the event that we add a posture image to a shoot that is not completely contained into the target shoot, this will distort the medial axis computation of the union of images and result in kinks the shoot curve.
Branching is detected on the individual pose level. For the medial axis of a single posture image, if the single medial axis curve diverges from a shoot, then it is said to be branching and a new child shoot should be formed.

Care must me taken in determining the parameters for when a curve is diverging. We define divergence distance to be the distance between the target curve endpoint and the diverging point on the parent curve. The diverging point is defined by the difference between the tangent angle of the point at which the curve last leaves the parent curve and the next point along the curve at which the minimum angle threshold has been made. The closest point on the parent curve to this minimum angle threshold point is the diverging point.

There are two constants that must be set that determine the threshold parameters for divergence detection. The first parameter is A, the angle difference between the departure point and the divergence point. The second parameter is D, the minimum distance threshold between the divergence point and the target curve endpoint.

The points are: departure point, divergence point, and end point.

7.7 Localizing On Shoot Map

Assuming that the decision of making a branch or not has been made by the branch detection method, and that all the current poses have been added to their respective shoots, we can assume that the current pose can be completely localized on some location of the shoot map. There are two possibilities for the locations of the posture curve. Either it is 1) completely contained within a single shoot or it is 2) overlapping two or more shoots at once. In the latter case, to overlap more than one shoot, the posture curve must be inside of a map junction.

Initially, the only information we have is the initial guessed location of the new pose from the chosen motion estimation method and the categorization of the pose into a particular shoot.

One use of the maps is to be able to localize the pose of the robot. Given the current state of the shoot map, we need methods by which we can localize the robot's pose.

Why should we want to localize the robot? Though we have used motion estimation techniques and local constraints to iteratively construct the map and robot's trajectory to the best of our ability, for circumstances when the robot is backtracking or returning to a location we have visited before, a localization technique will give us a way to perform loop-closing or data association. If we are traveling down a pipe in only one direction, loop-closing is not necessary since there is no chance of visiting the same location twice.

In the event of visiting the same location, there are two questions that must be answered: 1) is the robot at a location that it has visited before and 2) if so, where is its best fit? Since we have already decided if this pose is branching and have added all the necessary branch points, and categorized this pose into individual shoots, it is already true that this pose is somewhere on the map. However, the location that we chose for the pose may be incorrect or may have broken the map.

1) Cases where the pose is in the wrong location: - Poorly overlapped poses add kinks to shoot - Many locally identical indistinguishable locations - Shoot map self-intersects

If the current pose is wrong, but all past poses are correct, then it is a simple matter of fixing the current pose. However, as is often the case, if a whole history of poses is incorrect, it becomes more challenging to fix an entire history of poses. We have an approach to selecting different histories of poses in the next chapter on Probabilistic Mapping, but for this section we will only focus on fixing the current pose.

7.8 Finding the Best Fitted Location

Immediately after motion estimation, branch detection, and shoot categorization, our next task is to find the best likely location for the current pose. We measure the likelihood in terms of the best possible fit of the current posture curve to the current shoot map. To do this, we again utilize ICP between the posture curve and splice curves representing different permutations of spliced sections of the shoot map.

For instance, if we wanted to test whether the posture curve fits into a junction shown in Figure X, there are three different spliced curve permutations of this junction. By performing ICP of the posture curve onto each of these splice curves, we receive different candidate localized poses for the robot's current position. By performing several such ICP experiments with several splice curves, we can apply a selection criteria to be our best fit, use that as our maximum likelihood pose, and make any necessary changes to the shoot map.

We now describe the steps we take during the localization phase:

1) Choose the shoot splices that we wish to localize on 2) Select a set of initial poses for each splice to input to our ICP algorithm 3) Perform ICP on each initial pose on each splice 4) Collate, sort, and filter results 5) Choose most likely result and set as the current pose.

7.8.1 Choosing Shoot Splices

In step 1), if there is only one shoot in the shoot map then the resultant splice is simply the parent shoot itself. However, if there is more than one shoot, a number of possible splices are possible. We can take an exhaustive approach and select all possible splices, including splices with multiple junctions contained within them.

One approach we can take is to select all splices that terminate at every possible combination of terminal points in the shoot map. So for instance, for a map with N shoot and N–1 junctions, the number of possible splices is represented by (2 + N - 1) choose 2. However this can be problematic if we consider junctions that are not in the neighborhood of the pose in question. In particular, if we have three junctions A, B, and C, there will be (4 choose 2) = 10 possible splices between all possible terminals. However, if only A is in the neighborhood of our pose, many of the possible splices are locally identical so there is no value in using splices that include junctions B and C unless B and C are in the neighborhood of the pose.

We can discriminate the type of splices we will use by excluding some junctions from consideration if they are not in the neighborhood. If the pose of P is less than some distance D to the branch point J of the junction, then we can instead treat the junction J as a termination point. For instance, in figure X, if we turn junctions B and C into termination points for the number of splices and include A as a junction of interest, we result in (3 choose 2) = 3 possible splices.

7.8.2 Selecting Initial Poses for ICP

Now that we have our set of shoot splices on which we will localize the current pose, we must determine the initial pose for each splice to input to our ICP algorithm. The ICP algorithm takes two sets of points and an initial transform between them and outputs another transform that represents the best possible fit. In our case, the two sets of points are uniform samples from the posture curve and the shoot splice curve respectively. The initial transform represents the initial pose of the posture curve and the output transform is the pose of the best locally optimal pose. Since the results are locally optimal, it makes sense to run the algorithm with multiple initial pose inputs to find a more globally optimal result.

Globally, there are multiple local minima that represent best fits of the posture curve. However, only one of these is the true location. We can bound the space of possibilities by only selecting initial poses that are near the initial pose from the motion estimation step. That is, we can select sets of input poses that are within a diameter of the initial pose P.

For each shoot splice, select the closest point to the initial pose P. Given a spacing of S_d, select new points along the curve such that the arc distance between them is S_d and the cartesian distance of P_s to P is less than some neighborhood diameter D_n. These represent the inputs to the ICP algorithm for each run. If we assume that we have 10 initial poses for each shoot splice and 3 shoot splices, we have 10x3 executions of the ICP algorithm and 10x3 output result poses.

7.8.3 ICP Algorithm

- Orientation
- Closest pairs intersection point
- Closest pairs computation
  - Local angle variance
  - Closest point A -> B
  - Closest point B -> A
  - Select pair that has lowest angular variance
- Result serves as intersection point between two curves.
- 3DOF -> 2DOF conversion
- (x,y,o) transform -> (u,o) transform
- Covariance and mahalanobis distance?
- Point-to-line constraint

- Point-matching search
- Cost function
- Repeat N number of times or convergence (c1-c2) $<$ minCostDiff

7.8.4 Collate, Sort, Filter

Given all of the results, we need to select just one as our final localized pose. We need some criteria to evaluate the fitness of each pose. There are a number of metrics that we can use in this evaluation. However, there is no straightforward method for choosing the best pose since there is ambiguity in the map, identically featured locations, and sometimes there is not enough discriminatory information to make a good localization fit. Therefore, the process by which we select a single pose to be our best fit is derived empirically and experimentally which we show here. The ambiguity and uncertainty of this problem is handled more explicitly in the next chapter on Probabilistic Mapping.

1. Curve extension
2. Curve divergence
3. Contiguity Fraction
4. Angle difference
5. Position displacement
6. Match count *
7. Overlap sum *
8. Utility function

utilVal = (0.5 - angDiff) /0.5 + (contigFrac - 0.5)/0.5 + (3.0- posDist)/3.0 + 1-(isExtension OR isDivergence)*0.5

Filter criteria:

- isExtension => Reject
- isDiverge => Reject
- contigFrac <= 0.5 => Reject
- posDist >= 3.0 => Reject
- angDiff > 0.7 => Reject

Sort by utility value Take result with highest utility

7.9 Curve Segment Algorithms

- IsOverlapped()
- getOrientation()
- getContiguity()
- GetOrderedOverlappingPaths()

The purpose of the divergence computation is to detect the concrete event of the target curve diverging from the host curve. This is used to classify whether or not a junction has been detected or whether the host path is simply being extended.

The difficulty associated with this task is that there is no discrete event to determine a divergence. It is measured by passing a threshold that must be calibrated to the task. The thresholds are both sensitive to cartesian distance and angular difference between the diverging curve and the divergence point on the host curve.

We first begin by computing the closest point distance between the target curve points and the host curve. At the point which the closest point distances begin monotonically increasing until the curve runs out, we start our search for the branch point. That is, we select the minimum distance inflection point before distance increases monotonically.

From this point, we walk.

Divergence between two curves is the point at which the overlapping curves separate. It is not a complete overlap but a partial overlap of two noisy curves. The challenge is defining under what conditions two curves are considered to be overlapping and under what conditions the curves are diverging. Finally, one must decide the location where the two curves transition from overlapping to diverging.

We may require the divergence point to have particular properties. That is, we may want the point to reflect an extrapolation of where the two curves would meet if they were not smoothed so much. We may also want to divergence point to have an angle that represents the approximate angle of divergence.

The divergence can be slow transition, but we would like to specify a fixed point at which it occurs. Satisfying the above conditions at the same time, determining the divergence point becomes a challenge.

We have defined 2 different types of divergence. The simple divergence and the angle-biased divergence.

The simple divergence is computed based on the closest point distance between the host curve and the target curve.

getDeparturePoint():

- receive as input the global path and the local spline of a pose

getOrderedOverlappingPaths()

The purpose of this function is to find the minimum fitting path splice for a given target local spline curve.

We define the minimum fitting splice as the shortest curve splice in which the target curve is completely contained.

For all possible combinations of path splices and for the target curve in its designated global pose, find the splice that has the highest contiguity and passes overlap condition for all of the splice's constituent path fragment.

The function returns an ordered list of path IDs that correspond to the overlapping order of the target curve on the selected path splice.

The purpose of the overlap condition computation is to determine if a proposed path splice curve is at all overlapping another curve such as a local spline or another path splice curve. The actual value of the overlap sum is not necessarily interesting, only that it is a tractable value instead of infinity.

It has two possible conditions: 1) is overlapping partially, and 2) does not overlap at all.

The computation is achieved by taking the target curve, which is usually the local spline, and for each point along the local spline, find its closest matched point on the secondary curve, usually the path.

The closest is also biased by the relative tangential angle between the two points. Two curves crossing each other's paths at 90 degrees will result in a non-overlapping condition since their are no points that have like tangential angles.

Also, closest points are limited by distance. Our upper value is 0.2 which is hand-calibrated constant. The value should be chosen based on the scale of the robot, the resolution of the sensor data, and the data noise and uncertainty. If a closest point less than 0.2 is not available, then no pair is created.

This particular computation is used as a quick and dirty filter for rejecting curve overlap hypotheses that do not overlap in any conceivable or workable way. It does not allow for any displacement or rotation of the two curves since this is the province of the ICP algorithms.

The purpose of this particular computation is to determine how much of two curves are contiguously overlapped. This is a complement to the overlap sum computation which computes scalar quantity which is an average of point-to-point closeseness over the whole curve.

The contiguity calculation determines what percentage of the target curve is completely overlapped and assumes that the remained of the portions are diverging from the host curve.

Contiguity is computed by counting the number of points on the target curve have a matched point partner with distance less than 0.2 whose neighbor's are also of distance less than 0.2. The value of 0.2 is experimentally chosen to reflect the type of data that we receive. If there are more than one set of contiguous points that are less than 0.2 distance to their matched pair, the larger set is chosen. Only the larget set is used in the percentage calculation.

The contiguity is computed by dividing the maximum total number of contiguous points with matched pair distance of less than 0.2 by the total number of points in the target curve. The value is from 0.0 to 1.0.

This particular metric is useful for validating divergence conditions. A non-diverging curve pair should have a contiguity of close to 1.0. Whereas, a target curve that internally diverges from the host curve or extends the host curve should have a contiguity of 0.7 or lower.

At what threshold to make decisions based on these unitless metrics is something that must be done experimentally, probabilistically, or in a machine-learning approach. The value itself is scale independent so it should not vary based on the size of the environment and robot, but on the relative data size of the compared curves.

The purpose of the overlap sum computation is to determine how well a proposed path splice curve overlaps another target curve such as a local spline or another path splice curve.

The computation is achieved by taking the target curve, which is usually the local spline, and for each point along the local spline, find its closest matched point on the secondary curve, usually the path. The sum of all the cartesian distances is added up and the result is the overlap sum divided by the number of points to give a normalized result.

If extraordinarily short curve overlaps well, it will have a comparable value to a very long curve overlapping well. It also introduces ambiguity for certain situations. If a curve overlaps at a medium constant displacement offset, its will value have the same result as a curve that half overlaps well and half overlaps poorly.

A fundamental primitive problem when mapping confined environments is determing when two partially overlapping curve segments diverge from each other. That is, at which point does curve A diverge from curve B. See the attached figure.

The definition of the divergence point is not obvious from the offset. Given that the two curves are not exactly overlapping and that the instance where curve A begins its divergence, it is not clear where the point should be defined. Generally, we want an extrapolation of where we think the intersection of two arms of a junction are intersecting. This requires the extrapolation of the incoming direction of the divergent curve and intersect to its parent curve.

We examine multiple approaches to this defiiniton.

Chapter 8

Probabilistic Mapping

8.1 Space of Possibilities

Given our shoot map representation in the previous chapter, our mapping decisions were based on a maximum likelihood evaluation of the pose's fitness. The best possible fit at the given moment is made and the decision remains committed. There is no opportunity to resolve ambiguity or correct decisions made in error. For some applications, this is sufficient. However, incorrect decisions can lead to irrevocable consequences in the map. Such things can include multiple repeated instances of a junction, misidentification of a junction, distortion of the shoots, and self-intersecting shoots.

Some examples of maps where the decisions of whether or not to branch or fit into an existing junction, and the accumulation of pose error over time, are shown in Figure X. (Y junction, 3-pass new branches) (T-junction backtracking fail)

We wish to broaden the number of scenarios that our mapping can handle. We wish to explicitly represent different possible valid maps given the input data. We need some way to explicitly represent the variety of different mapping choices and weigh them in terms of likelihood. Following the Bayes' Rule formalization of the SLAM problem, we first identify the sensor percepts, the robot actions, and the state space representation of the environment.

In our formulation, there are three possible actions that can be taken. Either the robot moves forward, it moves backward, or it moves not at all. The total distance traveled is not subject to control in our formulation, so there are only 3 possible actions. The robot simply takes a step. The series of actions is denoted by:

$$U_{1:t} = \{\text{fore}, \text{back}, 0\}_{1:t} \tag{8.1}$$

The percepts are represented by the posture images we created from Chapter 2. The percepts are represented as:

$$I_{1:t} = \{f(i,j) \text{for} 0 \leq i \leq N_g, 0 \leq j \leq N_g\}_{1:t} \text{where} f(i,j) = 0,1 \tag{8.2}$$

Starting from Chapter 6, we represented the map in terms of poses of each posture image in the environment. Representing and improving the map was approached in terms of finding better poses to make the map the most consistent. Different techniques were used to approximate motion and localize the pose based on previous poses. The poses are represented with the following notation:

$$X_{1:t} = x_t, y_t, \theta_t \tag{8.3}$$

With the introduction of the shoot map representation in Chapter 7, we have add the shoot structure composed of pose node membership set and a branch point pose.

$$S_{1:N} = \{b_i, c_i \text{for} 0 \leq i \leq N\} \tag{8.4}$$

$$c_i = \{\exists k \text{s.t.} 0 \leq k, k \leq t\}, |c_i \geq 1, \text{for} i = 1 \text{to} N \tag{8.5}$$

$$b_i = (x_i, y_i, \theta_i), \text{for} i = 2 \text{to} N, b_1 \text{isundefined} \tag{8.6}$$

We can begin to gradually introduce a probabilistic framework by defining the state space representation for each variable, representing it with a random variable, and approximating the Bayes' Rule.

8.2 Pose Particle Filter

Population of pose hypotheses, localized, evaluated, resampled.

We first begin by building and representing the state space of possible poses of the robot. The full and complete state space is the full space of possible (x, y, θ) combination values. Either a parameterization or discretization approach is required to computationally represent it. However, computing over the full range of values is computationally expensive and unnecessary. In our case, we make some assumptions and simplifications to make computation more tractable.

First, we take the maximum likelihood approach of all past poses to build our shoot map. However, the current pose is represented by a random variable and a belief distribution given the maximum likelihood map. We begin our probabilistic map by considering the multiple possible poses of the robot and increase the belief of poses that have the best possible fit.

Since the robot is in a confined space and all the sensed area is void space, it stands to reason that any possible location of the robot will be close to or on one of the shoot curves. With this observation, it is possible to reduce the dimensionality of the possible pose locations and make the localization problem much simpler.

The pose is originally represented by 3 values: (x, y, θ). We can reduce the dimensionality of the pose by restricting the pose to be a point on a shoot splice. Specifically, given a curve, a pose on the curve can be represented by two variables, the angle θ and the arc distance u. Therefore, the current pose of the robot given the maximum likelihood map is represented by the vector (u, θ) and its designated shoot splice s curve.

The shoot splice s is a small number of possible values. The 2 dimensional pose vector dramatically reduces the state space of the pose. Therefore, our posterior distribution need only cover 2 dimensions for a finite number of shoot splices. This makes the computational problem much more tractable compared to the full 3 dimensional state space. **Image of 3d space vs shoot splice space**

This approach assumes that any possible pose will be located on a shoot splice curve. This approach can encounter difficulties if this is not the case. Any situation where the pose would not be on the shoot curve is either, the shoot is distorted by improperly localized previous poses, or the robot is not in the center of the pipe. The latter case is possible if the pipe is too wide for the robot to anchor to the edges. One of our original assumptions is that environment is not too wide for the robot to anchor.

The choice of Bayes' Rule approximation method must consider a number of factors. These include the computational cost of evaluating a pose's fitness for a particular location, the computational representation of the prior and posterior, and the computation of the update for the Bayes' Rule equation.

$$P(X_{t+1}|X_{t:1}, U_{t:1}, I_{t:1}) = P(X_{t+1}) \times P(X_t|X_{t+1})/P(X_t) \tag{8.7}$$

8.2.1 Particle Filter Implementation

Particles represent a hypothesis of $u, \theta, s)$.

Update step.

1. Move step. Displace by arc distance + random(), align angle only (costly)
2. ICP fit, localize to all local splices. (costly)
3. For each particle, evaluate every splice localization (costly)
4. For each particle, select best fit
5. Create probability distribution over particles
6. Resample particles according to probability distribution

However, for the current pose, we represent the posterior of possible poses with a particle filter \cite{particle filter}.

8.3 Pose + Branch Location Particles

Pose + Small grid of branch locations. Evaluated by overlapCost of full medial axis of child on parent.

The next parameter of the shoot map that we wish to represent as a belief distribution is the branches and their branch locations. Specifically, we assume that whether a branch exists is a given and is taken as completely true for a particular map. The only question that remains is at what point does a new shoot branch from a parent shoot.

Given a parent shoot A and a child shoot B, what point on the shoot curve A does the shoot B branch from. A related question is, at what angle does shoot B branch at from the branch point on shoot A? The branch point can be uniquely described as a 3-tuple: (x, y, θ). Similar to section 8.2, we can reduce the problem state space by only considering points on the curve of shoot A. Therefore, a branch point can be uniquely described with the 2-tuple: (u, θ).

Our approach is to discretize the u-space over the parent shoot curve, making a finite number of possible branch locations. Choosing the correct granularity is critical. If the discretization is too sparse, it is possible that the correct branch location is missing from the possible locations. If the discretization is too dense, it will be too computationally costly to evaluate all possible locations.

An example of the discretized space over branch locations is shown in Figure X. Here the entire child shoot medial axis graph is shown with a weighted rendering corresponding to the location's fitness. Locations that overlap with the parent shoot are more likely. Locations where the child and parent don't overlap are evaluated as unlikely.

It is also clear from Figure X that there are is a large swath of possible locations that are not worth considering. For instance, the entire range of branch locations where the overlap is zero have virtually zero chance of being correct. It would be helpful if we could disregard this locations so that we need not continue to evaluate them during our computation. This can save on computational costs.

However, the question still remains, what set of possible branch locations *should* we keep track of? It helps to consider where information about branch locations come from in the first place.

Once a branching condition has been detected by a particular posture image and pose, a new branch point is created with respect to the pose. Therefore, branch points are conditionally dependent on poses. For a particular pose hypothesis, we must create a range of possible branch point hypotheses. That is, from pose X, the branch point detected at X' is located on the curve at u'. Therefore, we track a range of discrete branch point values around u'. In our implementation, we keep track of 11 possible values for a given pose hypothesis. We keep track of 40 pose hypotheses in the particle filter. This creates 11*40 possible branch points. Many of these branch points are identical.

8.4 Branch Decision

Shoot map for each go/nogo branch decision. Eliminate options based on degenerate conditions.