EXPLORING AND MAPPING CONFINED AND SENSOR-CHALLENGED ENVIRONMENTS WITH PROPRIOCEPTION OF A ROBOTIC SNAKE

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)

January 2013

Dedication

to ...

Table of Contents

List of Figures

List of Tables

Preface

Abstract

In many real-world environments such as flooded pipes or caves, exteroceptive sensors, such as vision, range or touch, often fail to give any useful information for robotic tasks. This may result from complete sensor failure or incompatibility with the ambient environment. We investigate how proprioceptive sensors can help robots to successfully explore, map and navigate in these types of challenging environments.

Our approach is to use a snake robot with proprioceptive joint sensors capable of knowing its complete internal posture. From this posture over time, we are able to sweep out the free space of confined pipe-like environments. With the free space information, we incrementally build a map. The success of mapping is determined by the ability to re-use the map for navigation to user-directed destinations.

We address the following challenges: 1) How does the robot move and locomote in a confined environment without exteroception? 2) How is the distance traveled by a snake robot measured with no odometry and no exteroception? 3) How does the robot sense the environment with only proprioception? 4) How is the map built with the information available? 5) How is the map corrected for visiting the same location twice, i.e. loop-closing? 6) How does the robot navigate and explore with the constructed map?

In order to move through the environment, the robot needs to have a solution for motion planning and collision-reaction. In an exteroceptive approach, we would detect and avoid obstacles at a distance. If collisions were made, touch sensors could detect them and we could react appropriately. With only proprioceptive sensors, indirect methods are needed for reacting to obstacles. A series of motion methods are used to solve these problems including compliant locomotion, compliant path-following, safe-anchoring, stability assurance, slip detection, and dead-end detection. We show results demonstrating their effectiveness.

While moving through the environment, the snake robot needs some means of measuring the distance traveled. Wheeled robots usually have some form of shaft encoder to measure rotations of the wheels to estimate distance traveled. In addition, range or vision sensors are capable of tracking changes in the environment to estimate position of the robot. GPS is not feasible because of its unreliable operation in underground environments. Our proprioceptive approach achieves motion estimation by anchoring multiple points of the body with the environment and kinematically tracking the change in distance as the snake contracts and extends. This gives us a rudimentary motion estimation method whose effectiveness we measure in a series of experiments.

Some method is needed to sense the environment. In an exteroceptive approach, vision and range sensors would give us a wealth of information about the obstacles in the environment at great distances. Touch sensors would give us a binary status of contact with an obstacle or not. With only proprioceptive joint sensors, our approach is to kinematically compute the occupancy of the robots body in the environment and record the presence of free space over time. Using this information, we can indirectly infer obstacles on the boundary of free space. We show our approach and the sensing results for a variety of environmental configurations.

Combining the multiple snapshots of the local sensed free space, the robot needs a means of building a map. In the exteroceptive approach, 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. The opportunities for spotting landmark features between consecutive views are limited. Our approach is to use the pose graph representation for map-building and add geometric constraints between poses that partially overlap. The constraints are made from a combination of positional estimates and the alignment of overlapping poses. We show the quality of maps in a variety of environments.

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. We instead develop an approach that exploits the properties of confined environments and detects loop-closing events by comparing local environmental topologies. We show the results of loop-closing events in a variety of environmental junctions.

Once we have constructed the map, the next step is to use the map for exploration and navigation purposes. In the exteroceptive case, navigating with the map is a localization problem, comparing the sensed environment with the mapped one. This is also the approach in the proprioceptive case. However, the localization accuracy along the length of a followed path is more uncertain, so our path-following algorithms are necessarily more robust to this eventuality. In the exteroceptive case, the act of exploration can be achieved by navigating to the boundaries of the map with no obstacles. In the proprioceptive case, exploration is achieved by

following every tunnel or path until a dead-end is detected. We show some environments that we successfully map and navigate, as well as some environments that require future work.

Chapter 1

Introduction

1.1 Problem

Environments such as pipes and caves are difficult to robotically map because they are opaque and confined. An environment is opaque when external range and vision sensors are inoperable because of lighting conditions, environmental fluids, or tight proximity to obstacles. An environment is confined when a robots mobility is restricted by the close proximity to multiple obstacles.

A solution to this problem would allow mapping and exploration of previously inaccessible environments. An approach that is invariant to ambient fluid (air, water, muddy water) 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.

These environments often render range sensors ineffective due to occlusions, distortions, reflections, and time-of-flight issues. Changing visibility and lighting conditions will impact the usefulness of vision sensors. Hazardous environments may also damage fragile external touch sensors. This forces us to rely on a strictly proprioceptive approach for building environmental information. Such sensors include accelerometers and joint sensors. This sensor information is by definition, highly local to the robots position and sets us at a disadvantage to other mapping approaches that traditionally build maps with large sweeps of range data. We have fundamentally less information with which to build maps and make exploration decisions.

An unstructured and irregular environment will require an agile robot with an equally agile locomotion approach. For instance, a wheeled robot may not be capable of traversing through a complicated cave artery or debris pile but a snake robot will be able to. Also, the geometry of the robot should not limit its ability to navigate. For instance, a spherical robot may be able to go through a pipe of sufficient diameter, but if there is a brief pinch in the passage, the robot will not be able to continue. The robot should be able to reposition itself to fit within the contours of the environment.

Global positioning information is also unavailable because of the enclosed nature of the environment. 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. Another method must be found.

1.2 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 to make use of vision and range sensors, its location underground makes the use of GPS technologies problematic.

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 robotic grasping literature and is described as contact detection. That is, the identification and reconstruction of object contacts from the joint sensors of the grasping end effector. These include Kaneko, Grupen, Haidacher, Mimura.

Finally, in addition to our early 2009 paper on this concept [Everist09], another researcher tackled the problem of mapping a confined and opaque environment of a pipe junction in an oil well bore hole deep underground [Mazzini11]. Mazzinis approach focuses on mapping one local area of the underground oil well and uses physical probing to reconstruct the local features while anchored to a single reference frame.

1.3 Approach

In this thesis, we use a snake robot in a physics simulation environment to develop a theoretical framework to map confined spaces with no external sensors. With our simulation running the Bullet physics engine, we create a number of flat pipe-like environments in which our robot will explore and map.

We choose a snake robot form as our mapping robot due to its flexibility in moving through tight spaces and the large number of joint sensors from which we can extract environmental information. By pushing against the walls on either side of the pipe to form anchors and pulling the body forward, we achieve snake locomotion with a variation on the biological snake concertina gait.

Though we do not simulate any sensor-inhibiting fluid in the environment or sensor-failure scenarios, we prohibit ourselves from using any type of external sensor, be it touch sensors, vision sensors, or range sensors such laser range-finders and sonar. Additionally, we do not allow ourselves to use any type of global positioning system since it is not practical to expect GPS to function underground. The only sensors we permit ourselves to use are joint sensors.

To build the features of the environment, our robot sweeps its body through the environment and captures the free space through its body posture. The free space becomes our dominant feature in map-making. We use the pose graph representation for map-making, where at each pose, we build a local free space map. We choose the geometric constraints between poses to ensure the correctness of the map.

We present several example environments, and we demonstrate the robots mapping and navigation results using our framework. Our metric for success for determining the quality of a map is measured by selecting a random location on the map and judging how well the robot reliably navigates to that location in the world.

Experimental System

We explicitly chose to develop our approach in a simulation environment in order to rapidly develop a theoretical framework to the problem of mapping without external sensing. In future work, we will apply these techniques to physical robots.

We choose to use the Bullet physics engine (version 2.77) because it is free, open source, mature, and actively developed. Though it is primarily designed for games and animation, it is acceptable to our robotics application. Our primary requisites are simulation stability and accurate modeling of friction and contact forces. True friction and contact modeling are not available to us without specially designed simulation software at a monetary and performance speed cost. Our choice of Bullet gives us believable results so long as we do not demand too much and try to simulate challenging situations (e.g. high speed, high mass, 1000s of contact points).

We focus our attention on environments such as the one depicted in Figure X. We build a flat plane and place a number of vertical walls to create a pipe-like maze environment. All environments we study are flat and have no vertical components. This means we need only focus on building 2D maps to represent the environment. From here on in this paper, we refer to such environments as pipes even though they may not correspond to the even and regular structures of physical utility pipes.

A snake robot is placed in the pipe environment as shown in Figure X. The snake robot consists of regular rectangular segments connected by actuated hinge joints. Each of the joint axes is parallel and coming out of the ground plane. This means the snake has no means to lift its body off the ground because all of its joints rotate in the plane of the ground. It is only capable of pushing against the walls and sliding along the ground. We choose a reduced capability snake robot because we wish to focus on the mapping problem instead of the more general serpentine control problem.

We can parameterize the snake and the pipe environment. For the snake robot, l is the snake segment length, w is the segment width, N is the number of segments connected by $N-1$ joints, and m is the maximum torque capable by each of the joint motors.

For the environment, we can begin by defining the pipe width W. For a pipe with parallel and constant walls, this is a straightforward definition as seen in Figure X. For non-regular features, we define the pipe width at a given point on one wall to be the distance to the closest point on the opposing wall. This correctly captures the fact that a robot that is wider than the smallest pipe width W will be unable to travel through that smallest width and will create a non-traversable pinch point. Conversely, a pipe width W that is larger than the reach of a robot will become a void space that the robot will have difficulty completely sensing without the aid of external sensors. For both the minimum and maximum pipe widths of a given environment, we define W_{min} and W_{max} respectively where $W_{min} \leq W_i \leq W_{max}$ where W_i is the pipe width at some point on wall p_i of the pipe environment.

Each joint on the robot is actuated by a motor and PID controller. It can actuate the joint anywhere from ±160 degrees. It uses the built-in motor capabilities of Bullet that allows us to set the joint angular velocity at run-time. Therefore, all outputs of the PID controller set velocity commands to the joint and the physics engine does its best to satisfy those as velocity constraints.

The structure of the PID controller is as follows:

```
err = target_phi - phi
If |err| > tolerance:
  errSum += dt*err
  errDiff = err-lastErr
  lastErr = err
  cmdVel = pgain*err +igain*errSum+dgain*errDiff/dt
```

```
if cmdVel > maxVel:
  cmdVel = maxVel
if cmdVel < -maxVel:
  cmdVel = -maxVel
```

Line 1 defines the error as the difference between the target and the actual joint angle. Line 2 prevents the controller from executing if the error falls below an acceptable threshold. This prevents the motor from attempting to perform minor corrections to an already near-correct angle in order to avert oscillations or error-producing compensations. Line 3 is the summation term for error over time while line 4 and 5 is the instantaneous error change from the previous controller iteration. The actual PID control law is shown on line 6 where P is the proportional term coefficient, I is the integration term coefficient, and D is the derivative term coefficient. The result outputs a command velocity for the Bullet engine. Finally, lines 7-10 limit this velocity to a maximum.

Each of the joints gives angle position information to simulate a shaft encoder or potentiometer. For this study, we do not simulate calibration error, sensor noise, resolution error, or gear backlash. Calibration error has been studied elsewhere /cite and there exists correction mechanisms for it. In the case of sensor noise, the noise on potentiometers is small enough not to affect our algorithms. Gear backlash was encountered and studied in /cite Mazzini. The primary error of concern is resolution error caused by the discretization process of a shaft encoder or an A/D converter. This problem has been studied /cite. Given a sensitive enough A/D converter (really?), this problem can be eliminated. In this study, we assume correct and noise-free joint sensors.

A joint provides an API to the controlling program with 2 read-write parameters and 1 read-only parameter. The read-write parameters are the target joint angle $^{j}_{i}$ and the maximum torque m_i, with the actual angle j_i being the read-only parameter. $^{j}_{i}$ is the angle in radians that we desire the joint to rotate to. J_i is the actual angle of the joint that reflects the current physical configuration of the robot. M_i is the maximum permitted torque that we wish to limit the individual motors to. The maximum torque can be lowered to make the joints more compliant to the environment.

1.4 Contributions

Our contributions in this dissertation are the following:

- Snake movement without external sensors
- Proprioceptive odometer
- Sensing with only proprioception
- Build map with poses of local profiles that are not remotely sensible
- Junction detection and loop-closing
- Navigation and exploration with a proprioceptive map
- Able to map and navigate several types of environments: Y-junction, T-junction, X-junction, L-junction, etc.

We present our contributions in each of the subsequent chapters.

Chapter 2

Snake Locomotion without Exteroception

In order to successfully control a snake robot and have it move through the environment, we need a solution for motion planning and reaction to collisions. Since we dont have exteroceptive sensors, this makes the problem challenging. The robot is unable to see what is right in front of it and must either move into open space or collide with obstacles.

Part of the challenge is having no external sensors and the other part is general task-oriented control of a hyper-redundant robot. There are many biologically-inspired locomotion strategies that live snakes in nature use to move throughout the environment. The closest biological gait solution to moving in confined environments is the concertina gait shown in Figure X.

This gait is characterized by a concertina motion of alternating extensions and contractions that result in a forward movement. The snake body pushes against both sides of the walls establishing anchors that allow the snake to alternate between pulling and pushing itself forward. Success of locomotion depends on the snake establishing high friction contacts with the environment with which to push and pull.

However, the differences between a real snake and our robot snake make a true implementation impossible. A real snake has the luxury of vision and olfactory sensors to provide early motion planning and a rich tactile sensing surface skin to provide feedback to its control system. Our robot snake has only the benefit of internal proprioceptive joint sensors. If we knew the exact width of the pipe walls, we could prescribe the perfect concertina motion to move through the environment smoothly.

A simple approach to making the fully anchored concertina pose, we could use the following equation:

$$jointCmd = A_J * cos(i * 2 * \pi * f_J/N)$$

where f_J is the frequency or the number of sinusoidal cycles per unit segment length, A_J is the maximum joint command amplitude, N is the number of segments on the snake, and i is the joint number we are on from 0 to N-2 since there are N-1 joints for N segments. This equation requires some tuning to give the right results and can result in postures shown in Figure X. In order to effect locomotion, a sliding Gaussian mask should be applied to the joint command output to transition parts of the snake between fully anchored to fully straight to reproduce the biological concertina gait.

The difficulty of this approach is that we need a priori knowledge of the pipe width, the configuration of the snake needs to be tuned by hand for the particular snake morphology, and there is no sensory feedback to this implementation, so no adaptive behavior is possible. We need approach that will work in environments of unknown and variable pipe width. Our robot needs to be able to adapt its locomotion to the width of the pipe.

Our desire is to be able to prescribe any type of snake pose, for any anchor width and any snake segment length or parameters, and have the snake automaticaly assume that pose. In order to do this, we need a means of specifying the pose we want and an inverse kinematics method for achieving that pose.

For both of these requirements, we use the backbone curve-fitting methodology first described by Chirikjian (cite). This method of control is to produce a parameterized curve that represents the desired posture of the snake over time. Over time the curve changes to reflect desired changes in the snake posture. Snake body curve fitting is achieved by finding the joint positions that best fit the snake body onto the curve. This is found either by direct calculation or a search algorithm.

An example of backbone curve fitting can be seen in Figure X. The parameterized curve is shown in light blue in the background. The blue snake body in the foreground is fitted onto the curve using an iterative search algorithm for each consecutive joint.

A typical application would be to specify the posture of the snake using a sinusoidal curve and then applying an inverse kinematics technique to fit the snake to the curve. The problem can be formulated iteratively as follows: for joints $i, 0$ whose poses $(p_i p_0)$ are on the curve C with parameters $t_i - t_0$, find the joint angle a_i such that $p_i + 1$ is on the curve C and $t_i + 1 > t_i$.

For the equation C defined as:

$$y = A * sin(2 * \pi * x)$$

the equation is monotonic along the x axis, so satisfying the criteria $t0 < t_1 < < t_i$ is the same as satisfying $x0 < x_1 < ... < x_i$.

If the length of snake segment is l, we specify a circle equation centered at the joint position (x_i, y_i) with the following:

$(x - x_i)^2 + (y - y_i)^2 = l$

Finding solutions for the simultaneous equations A and B will give us possible solutions to $x_i + 1$ and $y_i + 1$. For these two sets of equations, there are always at least 2 possible solutions. We need only select the solution that satisfies the invariant condition $x0 < x_1 < ... < x_i < x_i + 1$. There may be more than solution. In which case, we select the $x_i + 1$ where $(x_i + 1 - x_i)$ is the smallest. This prevents the snake from taking shortcuts across the curve and forces it to fit as closely as feasibly possible to the parameterized curve given the snake robots dimensions.

The above simultaneous equations do not have closed-form solutions and instead must be solved numerically. This can be computationally expensive using general equation solvers. Instead we propose a more specialized solver for this particular problem.

We choose a series of uniform samples $({}^{p}_{0}...{}^{p}_{k}...{}^{p}_{M})$ around a circle of radius l centered at (x_i, y_i) such that all points ${}^{p}_{k}$ have ${}^{x}_{k} >= x_i$. We are looking for instance of crossover events where the line segment $({}^{p}_{k},{}^{p}_{k}+1)$ crosses over the curve C. Given at least one crossover event, we do a finer search between $({}^{p}_{k},{}^{p}_{k}+1)$ to find the closest point on the circle to the curve C and accept that as our candidate position for joint point $p_k + 1$. As long as we assume that the curve C is monotonic along the x-axis and a point of the curve C can be computed quickly given an x-value, this is a faster method of computing intersections of the circle with the backbone curve than a general numerical solver.

For instance, in Figure X, we show a curve C described by a sine curve and a series of circles intersecting with the curve that indicate candidate locations for placing the subsequent joint on the curve. These are used to determine the appropriate joint angles to fit the snake onto the curve.

Now that we have a means of changing the snakes posture to any desired form given a parameterized, monotonic curve that describes the posture, we now need to form an anchoring approach that will work for arbitrary pipe widths.

2.1 Anchoring

Single sine period Some way to fit the anchor to the pipe width without direct sensing Use error model as our contact detection Joint error from command position Refined search

In order to fit the anchor points to a pipe of unknown width, we need some way of sensing the walls. Since we have no exteroceptive sensors, the best way of sensing the width of the walls is by contact. Since we have no direct contact sensor per se, we must detect contact indirectly through the robots proprioceptive joint sensors.

Using the backbone curve fitting approach, we take one period of a sine curve as the template form we wish the robot to follow. We then modify this sine periods amplitude, at each step refitting the snake robot to the larger amplitude, until the robot makes contact with the walls. This establishes two points of contact with the wall which assist in immobilizing the snake body.

It is clear from Figure X that as the amplitude of the curve increases, more segments are needed to complete the curve. If the sine curve is rooted at the tip of the snake, this results in the curve translating towards the snakes center of mass. Instead, we root the sine curve on an internal segment, as seen in Figure X, so that the position of the anchor points remain relatively fixed no matter the amplitude and the number of segments required to fit the curve.

Two points of contact are not statically stable if we disregard friction. Though friction exists in our simulation and in reality, we do not rely on it to completely immobilize our robot. Our normal forces are often small and the dynamic and transient forces can easily cause a contact slip. Therefore, we need at least 3 contact points to consider an anchor to be statically stable.

One way to achieve this is by adding new sine period sections to create another pair of anchors. The amplitude of the new sine period is separate from the previous anchors amplitude and is represented by a new curve attached to the previous one. This way the amplitudes remain independent. The equation to describe two sets of sine period curves with independently controlled amplitudes are as follows:

$$u(t) * u(2\pi - t) * A1 * sin(f * t) + u(t - 2\pi) * u(4\pi - t) * A2 * sin(f * t)$$

Or to generalize for N periods and N pairs of anchor points.

$$SUM_i = 0_N : u(t - i * 2\pi) * u((i + 1) * 2\pi - t) * Ai * sin(f * t)$$

So now that we have the means to control the amplitude of our anchor points, we need some means of detecting that a contact is made and another to make sure the anchor is secure.

Our approach is to gradually increase the amplitude of an anchor curve until contact with both walls has been made. We will continue to increase the amplitude until we see significant joint error occur when fitting to the curve. If the amplitude because larger than the width of the pipe, the snake will attempt fitting to a curve that is impossible to fit to and will experience a discrepancy in its joint positions compared to where it desires them to be.

The structure of this error will normally take the form of one or two large discrepancies surrounded by numerous small errors. This reflects that usually one or two joints will seriously buckle against the wall, while the rest of the surrounding joints can reach their commanded position without disturbance. An example of an anchor curve with larger amplitude than the width of the pipe is shown in Figure X.

The contact is flagged once the maximum error reaches a certain threshold across all the joints of the anchor curve. Of the joints M of the anchor curve and the error e_j, if there exists $e_j > 0.3$ radians, than we flag a contact. It isnt so much that we are detecting a contact but a minimum pushing threshold against the wall. The robot is pushing hard enough to cause joint error over an experimentally determined threshold.

If j_k through $j_k + M$ are the joints comprising the fitted anchor curve, the error of one joint is defined by $e_k = |theta_k - theta_cmd|$. The maximum error is defined by $maxError = max(e_k, , e_k + M)$.

The amplitude is initially changed by large increments to quickly find the walls. Once the threshold has been reached, the current and last amplitude are marked as the max and min amplitudes respectively. We then proceed to perform a finer and finer search on the amplitudes between the max and min boundaries until we reach a snug fit that is just over the error threshold. This is a kind of numerical search(?).

The pseudocode for this algorithm is as follows:

```
currAmp = 0
minAmp = 0
maxAmp = Inf
ampInc = 0.04

while ( maxAmp-minAmp >= 0.001 and ampInc >= 0.01 )
  currAmp += ampInc
```

```
isError = setAnchorAmp(currAmp)

if ( !isError )
  minAmp = currAmp
Else
  maxAmp = currAmp
  ampInc /= 2
  currAmp = minAmp
  setAnchorAmp(currAmp)
```

The main objective of this code is to reduce the difference between maxAmp and minAmp as much as possible. minAmp and maxAmp are the closest amplitudes under and over the error threshold respectively. The function setAnchorAmp() sets the amplitude, performs the curve fitting, and reports back any error condition as described earlier. Once this algorithm is complete, we have made a successful anchor with two contact points under compression without the fitted anchor curve being distorted by joint buckling.

, we must determine if the anchor is secure. That is, if we exert some amount of transient force on the anchors, will the anchor slip?

First we must define what makes an anchor secure.

Jerk Test

2.2 Curves

When we describe curves for use in backbone curve fitting, they must be assigned to a frame of reference and usually have a starting point. Most often, the frame of reference chosen is one of the local segment frames on the snake body. However, sometimes we could choose the global frame if we wanted to navigate or probe something specific in the environment. The local frame curves that we use all start from a specific body segment and sprout from that segment like a plant. We say that this curve is rooted in segment X.

2.3 Behaviors

Rationale: Allow separate behaviors to run on different parts of the snake, merge their functionality to create composite behaviors (subsumption architecture) Apply Mask for Smooth Transition Define behavior. Purpose is fulfill a type of task. Takes inputs and produces motor outputs.

Parent-child relationship. Childs instantiated and destroyed. Behaviors in parallel. Sometimes need conflict resolution for trying to control same joint. Either priority-merging or splice-merging, parent is responsible. Outputs of child behaviors may be subsumed by parent behavior.

Our method of control is a behavior-based architecture that is charged with reading and tasking the servo-based motor controllers of each joint. Each behavior may be a primitive low-level behavior or a high-level composite behavior composed of multiple sub-behaviors. Furthermore, a behavior may have complete control of every joint on the snake or the joints may be divided between different but mutually supporting behaviors. This architecture allows us to achieve a hierarchy of behavior design as well a separation of responsibilities in task achievement. For instance, the back half of the robot could be responsible for anchoring, while the front half could be responsible for probing the environment as seen in the example architecture in Figure X.

Each behavior in the behavior-based architecture is time-driven. It is called periodically by a timer interrupt to compute the next commands for the following time-step. This could be variable time or constant time, but to make our behavior design simple, we use 10ms constant time steps in our simulation.

At each time step, the state of the robot is captured and driven to the behaviors. The data includes a vector of joint angles $theta_i$, a vector of commanded angles phi_i, and a vector of maximum torques m_i. These values were described in the previous section X. The output of each behavior is a vector of new commanded angles $hatphi_i$, a vector of new max torques $hatm_i$, and a vector of control bits c_i. The values of $hatphi_i$ can be any radian value within the joint range of motion or it can be NULL. Likewise, the new max torque of $hatm_i$ can be any non-negative max torque threshold or NULL. The NULL outputs indicate that this behavior is not changing the value and if it should reach the servo-controller, it should persist with the previous value.

The bits of the control vector, c_i , are either 1, to indicate that the behavior is modifying joint i, or 0, to indicate that it has been untouched by the behavior since its initial position when the behavior was instantiated. This control vector is to signal the activity to parent behavior that this part of the snake is being moved or controlled. The parent behavior does not need to respect these signals, but they are needed for some cases.

Our behavior-based control architecture follows the rules of control subsumption. That is, parent behaviors may override the outputs of child behaviors. Since we can also run multiple behaviors in parallel on different parts of the snake, sometimes these behaviors will try to control the same joints. When this happens, the parent behavior is responsible for either prioritizing the child behaviors or adding their own behavior merging approach.

Asymmetric behaviors that run on only one side of the snake such as the front are reversible by nature. That is, their direction can be reversed and run on the back instead. All behaviors with directional or asymmetric properties have this capability..

Finally, parent behaviors can instantiate and destroy child behaviors at will to fulfill their own control objective. All behaviors are the child of some other behavior except the very top root-level behavior which is the main control program loaded into the robot and responsible for controlling every joint. The root behavior is responsible for instantiating and destroying assemblages of behaviors to achieve tasks as specified in its programming. We will discuss some of these behaviors and child behaviors in the following sections.

2.4 Smooth Motion

Smooth Motion Rationale: do not upset the anchors No jerky motion that cause transient dynamics Collisions will not make anchors slip Step-based motion and linear interpolation between positions HoldTransition behavior

We desire the motion of our snake robot to be slow and smooth in nature. It does us no benefit for the motion to be sudden and jerky. Moments of high acceleration can have unintended consequences that result in our anchor points slipping. Either collisions with the environment or transient internal dynamics can cause the anchor points to slip.

Our method of ensuring smooth motion is to take the initial and target posture of the joints of the snake, perform linear interpolation between the two poses, and incrementally change the joint angles along this linear path. For instance, if the initial joint angles are $s_0 ... s_n$ and the target joint angles are $t_0 ... t_n$, the interpolated path for each joint j is found by $g_j r = s_j + (t_j s_j) * (r/100)$ for $r -> 0, 100$, for 100 interpolated points on the linear path. The resultant motion is a smooth transition from initial to target posture as seen in Figure X.

This is the definition of the HoldTransition behavior. When it is instantiated, it is given an initial pose vector S and target pose T, where some values of t_i in T and s_i in S may have NULL for no command. At each step of the behavior, using the equation $g_j r = s_j + (t_j s_j) * (r/100)$, r is incremented to compute the new joint command. If t_j is NULL, then $g_j r = s_j$. Once $r = 100$, it returns True on step() and on all subsequent steps, $g_j = t_j$.

2.5 Behavior Merging

Behavior Splicing Rationale: Allow separate behaviors to run on different parts of the snake, merge their functionality to create composite behaviors (subsumption architecture) Apply Mask for Smooth Transition HoldSlideTransition

Precedence Merge Splice Merge Convergence Merge Compliance Merge

There are a few methods for merging behaviors that overlap in some way. One way of the previous section is for two behaviors to overlap temporally and using the HoldTransition behavior to handle a smooth transition from one configuration to another.

In the event that two behaviors control adjacent sets of joints or overlap their control in some way, a conflict resolution is method is needed to not only select the proper command for contested joints, but to prevent either behavior from negatively affecting functionality of the other by modifying any positional assumptions or introducing any unintended forces or torques to the other side.

A common occurrence is that two behaviors may try to control the same set of joints at the same time or otherwise. This is a conflict that requires resolution by the parent behavior. In this section we present three different ways to merge the outputs of different behaviors as means of conflict resolution. They are, in the order presented, the precedence merge, the splice merge, the convergence merge, and the compliance merge respectively.

Given that all child behaviors have an ordering, the default resolution method is to take the first behaviors control output over all others. This is called precedence merging. An ordering is always specified by the parent at the time it instantiates the child behaviors. If the first behaviors output is NULL, then it goes to the next behavior and so on until a non-NULL value is found or the last behavior returns a NULL, in which case the parent also returns a NULL unless the parent otherwise specifies.

Precedence merging can be crude when trying to maintain a stable and immobilized position in the environment. Sudden changes at the behavior-boundary can cause jarring discontinuities on the snakes posture and result in loss of anchoring as seen in Figure X. A more finessed approach would disallow these discontinuities and provide a non-disruptive merging. The first approach that does this is the splice merge.

In the event that two behaviors overlap, we have the option of having a hard boundary between the behaviors centered on a single joint. Except in the rare case that both behaviors want to command the splice joint to the same angle, it is highly likely that the merge will be discontinuous. In order to ensure that positional assumption is maintained for both behaviors, the behaviors must be spliced so that neither behavior disrupts the other.

If the splice joint is j_s, behavior A controls joints j_0 through j_s, behavior B controls joints j_s through $j_N - 1$, $S_s - 1$ is the body segment connected to j_s in behavior A, and S_s is the body segment connected to j_s in behavior B, we have the situation shown in Figure Xa. If behavior A commands j_s to phi_a to put segment S_s in the position shown in Figure Xb and conversely, if behavior B commands j_s to phi_b to put segment $S_s - 1$ in the position shown in figure Xc, the two behaviors can be spliced discontinuously with no disruption to either behavior by setting j_s to $phi_s = phi_a + phi_b$. The result is shown in Figure Xd.

This is the behavior splicing approach because it stitches the behaviors together in a clinical way to ensure proper functionality of both. Its primary role is to connect behaviors with discontinuous conflicts on the behavior boundary and whose positions must be maintained for functional purposes. For more fuzzy continuous boundaries, we use the convergence merging approach.

For the two behaviors shown in Figure Xa and Xb, the resultant convergence merge is shown in Figure Xc. For this approach, there is no discrete boundary but one behavior converging to another over several joints. For a given joint j_i somewhere in this convergence zone, and the commanded values for behaviors A and B being phi_a and phi_b, the converged command value for j_i is computed by:

$$phi_i = phi_a + (phi_b phi_a)/(1 + exp(i - C))$$

where C is a calibration parameter that controls the magnitude and location of the convergence along the snake. As $C- > +Inf$, $phi_i- > phi_b$. As $C- > -Inf$, $phi_i- > phi_a$. In practice, $|C| < 20$ is more than sufficient for complete convergence to one behavior or the other. The i parameter in the exponent ensures that the convergence value is different for each of the neighboring joints. This produces the effect shown in Figure Xc.

If we modify C, this gives us a control knob on the merge and can be moved up and down. If we move $C->+Inf$, behavior B takes complete control of the snake. If we move $C->-Inf$, behavior A takes control of the snake. C can be changed over time to create a sliding transition effect with a convergence boundary. In fact, this is the basis for a behavior called HoldSlideTransition.

A fourth and final way for two behaviors to be merged is the compliance merge. This is the case where two behaviors are not adjacent but are far away from each other, but still interfere with each other somehow by inflicting unwanted forces and torques between the behaviors.

A compliance merge resolves this by selecting a group of joints between the two behaviors joint sets to be compliant or non-actuated. This has the result of absorbing any forces or torques transmitted by either behaviors and mechanically isolating one side of the snake from the other. An example is shown in Figure X.

2.6 Compliant Motion

In the previous section, we mentioned compliance but not our implementation. Compliant motion is a much used technique in robotics and has been discussed widely (cite). The two approaches are either passive or active compliance. Our approach is to use passive compliance since we have no means to detect contact or sense obstacles.

Passive compliance is achieved by changing the settings on the max torque parameter of each joints servo controller. We use a high setting for active motion, a low setting for compliant motion, and zero for keeping the joint unactuated.

Besides using a compliance merge to mechanically isolate two behaviors, we also use compliance as an exploration strategy. Since we have no exteroception, the only way to discover walls and obstacles is to collide with them. Compliant impacts with obstacles have a couple benefits. It prevents the force of the impact from transmitting to the body of the snake and possibly causing anchor slip. It also allows the joints of the snake to give out and take the shape of the environment around it.

This latter benefit is one of our primary means of sensing and exploring the environment. By passively letting the snake robot assume the shape of the environment, it guides our mapping and our locomotion. Until we have mapped out open space and boundaries, we have no way of directing our robot towards them beyond colliding with the unknown.

2.7 Stability Assurance

In a previous section on smooth motion, we discussed an interpolation technique to smoothly transition from one posture to another. The primary motive of this was to keep the parts of the snake body that are immobilized from being destabilized by transient forces and causing anchor slip. Static forces from prolonged pushing against an obstacle or wall are a separate matter.

For a pose transition that has a sufficiently large number of interpolation steps of sufficiently large duration, we can reasonably be assured that no transient forces from dynamics or collision will cause anchor slip because we are moving very very slowly. However, sufficiently long transition times for reliability may be impractical for applications, since times for a simple transition can be upwards of 1-2 minutes using a conservative approach. For a robot that performs hundreds of moves that require transitions, this is an unacceptable approach.

If we focus on being able to detect transient forces on the snake body instead, we can use this to control the speed of our movements instead of a one-size-fits-all approach of long transition times for all motion. We have developed a heuristic technique to do just this.

Our approach is to use our only source of information, joint angle sensors, as make-shift stability monitors of the entire snake. That is, if all the joints stop changing within a degree of variance, we can call the whole snake stable. Once we have established that the snake is stable, we can perform another step of motion.

For each, we compute a sliding window that computes the mean and variance of the set of values contained in. If at every constant time step, we sample the value of some joint j to be $theta_t$, then we compute:

$$sMean = SUM(t-K, t, theta_i)/K$$

$$sVar = SUM(t-K, t, (theta_i sMean)^2)/K$$

K is the width of the sliding window or the sample size used for computing the variance. This and the time step size dT, are used to determine the resolution and length of variance computation. In our implementation, with some exceptions, we use $dT = 1ms$ and $K = 20$. That is, for N joints, we compute the variance of each joints angle for the last 20 readings once every 1ms. This may be changed and retuned depending on the available computing resources, the resolution of the sensors, and the sensors sample rate.

Finally, we can determine if the snake is stable if all of the joint variances fall below a threshold S_v and stay that way for over time S_t. In our case, we set $S_v = 0.001$ and $S_t = 50ms$.

We are primarily interested in preventing anchor slip. Most often, it is not necessary to wait for the entire snake to hold still before moving. Instead, we can focus just on the joints in the neighborhood of the anchor points. By computing just the variances of the joints in the local neighborhood of the anchor points, we can determine local stability. Using local stability, we gain some confidence that our anchor points do not slip between motion steps.

In order to use this concept of local stability, we need to have some indication of which joints to monitor and which to ignore. Fortunately, the control vector c, output of the behaviors mentioned in a previous section, is just the kind of information we need to know where to direct our local stability attention. If the values of a control vector indicate 0, this means that these joints are not modified by the behavior and should remain stable before performing another move step. If the values are 1, these joints are actively modified by the behavior and we should not expect these joints to remain stable between move steps. The behavior prescriptively tells us how it should behave and the stability system processes it to accommodate.

2.8 Adaptive Step

Show the behavior sequence in high-level terms. Show graphics.

Explain each behavior step in terms of the concepts we already learned.

Using these tools at our disposable, we can now build a suitable locomotion behavior for an unknown pipe with no exteroception. The entirety of our designed behavior is shown in Figure X and is called the adaptive concertina gait, a biologically-inspired gait based on the concertina gait but modified for the change in sensors. The behavior is divided into 6 states, and each level of Figure X shows one of those states. We describe the states as follows:

1) Rest-State (Initial): Rest-State is the initial fully anchored posture of the snake robot. This is our initial pose and it acts as a stable platform from which exploratory or probing behaviors can be executed to sense the environment with little chance of slips or vibration due to the multiple anchor contact points. There is no other action here than to hold the posture that it was already in. The Rest-State is the initial state and the final state of the concertina gait.

2) Front-Extend: In the Front-Extend state, the forward segments are gradually transitioned from their initial position to a straight-forward position. The joints are compliant to the boundaries of the environment to accommodate turns in the pipe.

3) Front-Anchor: The Front-Anchor state takes the extended portion of the snake and attempts to establish an anchor to the pipe as far forward as possible. The locomotion distance is maximized the further forward the front anchor is.

4) Back-Extend: The Back-Extend state follows a successful Front-Anchor event. All anchors established in the back segments are gradually extended until they are straight. Again, the joints are compliant so that they can conform to the walls of the environment.

5) Back-Anchor: The Back-Anchor state involves establishing multiple anchors with the remainder of the body segments.

6) Rest-State (Final): Upon conclusion of the Back-Anchor state, a single step of the adaptive concertina gait is complete and the snake is now in the Rest-State. From here we can do probing of the environment or perform another step forward.

A successful transition through all the states results in a single step. Multiple steps are used to travel through the environment. We describe no steering mechanism for this locomotion gait because the robot has no means to sense the environment in front of it. However, the robots compliant posture will adapt to any junctions or pipe curvature and follow it.

We now describe the implementation of each of these states and their composition of behaviors.

2.9 Front-Extend

GRAPHIC Merge(FrontExtend HoldSlideTransition Transition, HoldPosition)

Convergence merge of 0.0 extension and anchored state. Precedence merge of FrontExetend and HoldPosition. HoldPosition holds back half while FrontExtend controls front. Passes extended state to HoldSlideTransition. This manages convergence merge and slide with a Transition behavior to handle each step.

From the fully anchored Rest-State, the goal of the Front-Extend stage is to release the front anchors and stretch the front half of the snake as far forward as possible without compromising the rear anchors. The further forward we can extend, the further the snake can travel in a single step.

If the extended snake arm makes a lateral impact, the walls will guide the movement of the snake but not otherwise hinder the behavior. In the event of an actual obstruction, the tips movement will stop. We are able to track whether the position of the tip hasnt moved if overtime and are able to set a flag to abort the behavior at the current posture.

Its implementation consists of 4 behaviors arranged as shown in Figure X. We explain each behaviors functionality.

HoldPosition: The purpose of this behavior is to output the same joint commands indefinitely. It is instantiated with a vector of joint commands, and it continues to output these joint commands until termination. Here, it is instantiated with the entire initial anchored pose of the snake and continues to output this pose. The behaviors output is gradually subsumed by the growth of the FrontExtend behavior in a precedence merge. HoldPosition is placed 2nd in the behavior ordering at the root level for this reason.

Transition: This behaviors purpose is to take an initial and final pose and interpolate a transition between the two, outputting the intermediate postures incrementally. The number of interpolated steps used to transition between the poses is set by the parent behavior. It is often a function of how much difference there is in the initial and final poses. The clock-rate is dependent on the parent behaviors. Once it reaches the final pose, it returns a flag and continues to output the goal pose indefinitely until reset or deleted. Here it is instantiated by the HoldSlideTransition behavior and reset each time it needs to make a new move command.

HoldSlideTransition: This behavior was first mentioned in the Behavior-Merging section. Its role is to use a convergence merge to transition from one posture to another. The convergence parameter C is incremented over time to slide the convergence from the front to the back of the snake, to smoothly transition from the fully-anchored posture to the back-anchored and front-extended posture.

For each change in C, the Transition behavior is set up to interpolate smooth motion between the two postures. Once the Transition behavior completes, C is incremented and Transition is reset to the new initial and final postures. C is changed from 8 to 20 while using while using the joint numbers as the indices in the convergence merge equation shown in X, assuming that the joint numbering starts at 0 at the front tip. An example of the HoldSlideTransition behavior in action is shown in Figure X.

FrontExtend:

2.10 Front-Anchor

Merge joint selected. FrontAnchorFit. Jerk-behavior stage

Move merge joint backwards in case of run out of segments condition. Precedence merge is used because extended joints are all 0.0, straight.

Jerk joints selected as first 4 behind the behavior boundary. Set to (0,0)-¿(-60,-60)-¿(0,0)-¿(60,60). Lead, non-anchoring joints are compliant when not in use.

Change amplitude by modifying parameters to FrontAnchorFit. FrontAnchorFit computes IK for correct joint positions to fit the curve. Output is put to HoldTransition behavior. Controls the anchor portion only plus the unused portion on the tip.

FrontAnchor performs the adaptive anchoring technique using 3 points of contact instead of 2π with 2.5π cosine curve. If amplitude commanded to FrontAnchorFit requires too many segments, flag will be raised and merge joint will be moved back by 2 and restart.

Jerk test will be performed to see if the anchor is secure. If not, it will restart and move the merge joint back 2 to not only give more joints but also to move the anchor points back towards the snake body.

HoldTransition maintains the previous state of the back half and doesnt move it unless its commanded by FrontAnchor. This is primarily the eating of the extended joints.

During jerk test, HoldTransition is bypassed and FrontAnchor commands the 4 jerk joints directly to make a sudden movement.

GRAPHIC Merge(FrontAnchor -¿ FrontAnchorFit, — V HoldTransition)

2.11 Back-Extend

Set all joints to left of merge joint to 0.0. Set to HoldTransition and go. Weaken these joints so that any impacts they make wont disrupt the FrontAnchor which is our only anchor.

Avoids closed chain interference for the next stage.

GRAPHIC Merge(BackExtend — V HoldTransition)

2.12 Back-Anchor

Weaken the joints not anchored/feeder joints.

Starting at merge boundary, create a 2π sine curve that connects to the FrontAnchor in a seamless manner. Use splice merge on this boundary to connect the behaviors together.

Once an adaptive anchor has detected collision and selected a snug fit, we do not perform a jerk test since there is no means to do this. The only free joints are the extended joints that are hanging in space waiting to be used up.

After each successful 2π adaptive anchor has been, select another and fit the amplitude until anchor. Continue ad infinitum until the last joint is used in an anchor curve.

Use BackConcertinaCurve as the curve.

Once a period of anchors has been established, save the position and dont recomputed the IK since it can be expensive in large amounts.

Chapter 3

Motion Estimation with only Proprioception

Chapter 4

Sensing with Proprioception

Chapter 5

Building a Map with a Pose Graph

Chapter 6

Place Recognition and Loop-Closing

Chapter 7

Navigation and Exploration

Chapter 8

Conclusion