DOING THINGS WITH THE NAO : LIKE LASERS AND STUFF

THESIS

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

MASTER OF SCIENCE (Electrical Engineering)

at the

NEW YORK UNIVERSITY
POLYTECHNIC SCHOOL OF ENGINEERING

by

Griswald Brooks

July 2015

Approved:

_____
Advisor

_____
Date

_____
Department Head Signature

_____
Date

Copy No. # 1

University ID: N17150539

VITA

Dec, 1983 ..... Born

Jun, 2001 ..... Graduated from Inter-Lakes High School.

Sep, 2001 ..... Enlisted into the US Navy to be trained as an Electronics Technician.

Dec, 2007 ..... Seperated from US Navy with Honorable Discharge.

Jan, 2007 ..... Entered the Lakes Region Community College as a Computer Technologies major.

May, 2010 ..... Received AS Degree from the Lakes Region Community College.

Jan, 2010 ..... Entered the Polytechnic Institute of NYU as a Computer Engineering major.

May, 2013 ..... Received BS Degree cum laude from the Polytechnic Institute of NYU.

Jan, 2014 ..... Entered the NYU Polytechnic School of Engineering as a Electrical Engineering major.

May, 2015 ..... Received MS Degree from the NYU Polytechnic School of Engineering.

Dec, 2065 ..... Deceased due to Robot Apocalypse.

ACKNOWLEDGEMENTS

Thank Khorrami

Thank PK, Brian, and Agraj.

Dedicate to Mom and Dad. I totally wouldn't have even gone to this school if not for my Dad.

ABSTRACT

DOING THINGS WITH THE NAO : LIKE LASERS AND STUFF

by

Griswald Brooks

Advisor: Dr. Farshad Khorrami

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science (Electrical Engineering)

July 2015

This thesis shows what I did over the course of two years. I did something with the Nao robot and made it move and look at things and stuff. You'll see.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

Introduction

1.1 Concept and Motivation

A common task for robotic systems is for a mobile platform to transport itself from one location to another in the presence of environmental obstacles. Within this general task there are many subproblems. The system must have some sort of awareness of the destination objective and usually some notion of where it is relative to that destination. The destination location and the location of the robot can be encoded in a map that the mobile platform or some off board system is responsible for maintaining. Within the environment there may be obstacles that prevent the robot from moving through that space which can also be encoded into the map. Given these notions of robot, goal, and obstacle locations often an initial path is planned to allow the robot to reach the goal location. In addition to these higher level concepts, the robot must have a scheme by which it transports itself through the environment be rolling, walking, flying and controls for each of those methods. Solutions to these subproblems, map building, robot localization, obstacle localization, path planning, locomoting and others work together to accomplish the overall goal. Each of these subproblems can be expanded upon and this thesis works on three of these components: local navigation, obstacle detection and characterization, and gaiting. Used to work on these problems was the Nao Humanoid Platform by Aldebaran Robotics.

1.2 Platform Overview

The broader task of moving an agent from one location to another is applicable to a wide range of robots but each platform will have details that change the scope of the problem and the method used to approach it. The three problems focused on (local navigation, obstacle detection, and gaiting) for a flying robot with a global camera system will require different solutions than a wheeled robot with only rangefinders. For this thesis, the Nao H25 V4 Humanoid Platform by Aldebaran Robotics was chosen to act as

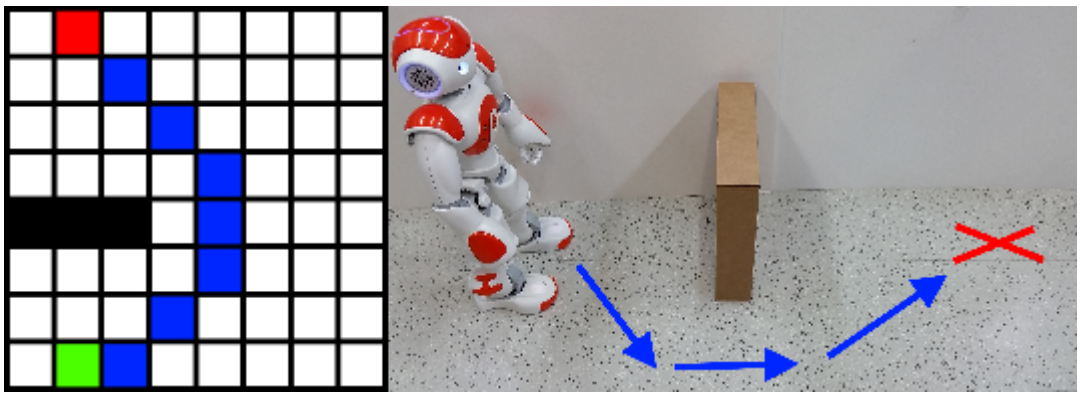

Figure 1: Example illustration of the Nao humanoid robot in an environment with an obstacle. The red X represents a goal location with the blue arrows showing an possible path. On the left side is one possible representation of the environment as a 2D grid.

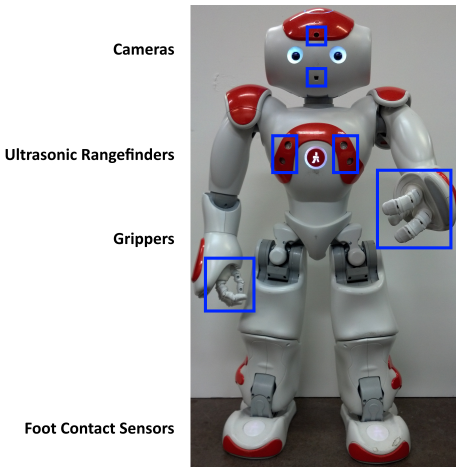

Figure 2: Coronal view of the Nao humanoid with a few pertinent features highlighted.

the mobile base. With the future goal of having robots interact in human environments, humanoid platforms have become an active area of research as they are physically compatible with such environments. While Nao is a 1.9 ft tall humanoid weighing 9.5 lbs and therefore only an appropriate analog for a young human toddler, the humanoid format is of primary importance to algorithmic development. with 25 degrees of freedom including two legs, two arms, and two grippers. Figure 2 shows a cursory illustration of the humanoid configuration. This lightweight but capable configuration enables research into mobile manipulation, humanoid gaiting, and terrain adaptation without the need for specialized support equipment such as belays or dedicated experimentation areas as the robot is safe for humans to interact with.

The robot comes with a suite of sensors including two cameras, two ultrasonic rangefinders, a 3-axis accelerometer and 2-axis gyro, foot contact sensors, and angular position encoders on every joint. Such a complement of sensors aids the robot in creating an estimate of a number of variables including the robot state and environment characteristics. Nao has a built-in WiFi radio and 1.6 GHz Intel Atom processor running a version of the Linux operating system allowing the robot to be programmed in C++ or Python using standard libraries that can be remotely uploaded. These features allow the use of a wide variety of tools when constructing new algorithms and for program iteration to happen at a rapid pace.

With all of these features the Nao makes for a convenient platform for the development of mobile robot algorithms.

To augment the sensor suite, a Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder (Lidar) was mounted to the Nao and interfaced via a USB connection to the Nao's onboard computer. The sensor has a scanning area of 240° with an angular resolution of 0.36° and a range of 5 meters. It is used to generate a planar point cloud representing the range to the nearest occlusion at a set of fixed bearing relative to the robot. Lidars are commonly used in mobile robotics so developing algorithms utilizing them is of practical concern.

1.3 Problem Domain

The three areas selected for development were local navigation, obstacle detection and characterization, and gaiting. Each of these is a broad research topic in its own right so the domain of investigation for each problem must be specified.

1.4 Implemented Components

1.4.1 Navigation

1.4.2 Crawlspace Detector

1.4.3 Crawl Gait

Someone else used a CPG network with an actor-critic scheme to generate a baby gait on a Nao and iCub.

1.5 Thesis Structure

This thesis is organized as follows:

CHAPTER II

Platform

For this thesis we used the Nao as the mobile platform. Aldebaran makes him. He's a cool little robot and he allows us to explore both things we wanted to look at which were navigation and gaiting. He's mobile, small, "cost-effective" and has a good API that we can use to do lower-level control of things when we want to, and abstract ourselves from it when we don't want to. The small size means he's easy to work with.

There's definitely some more opening paragraph to be written here about the Nao. Probably say something about why he's useful to investigate crawling. No one's going to send Nao into a disaster zone or anything but he's not to far off from robots that you would send in and we don't have to spend all the money or build a new lab to work with him. You can do lot's of simulations, but in the end you still have to test on a real robot.

While the Nao is cool and all, the sonar sensors just don't cut it for what we want to do. Lucky, Nao comes with a USB port and uses x86 and linux so it's relatively straightforward to add new things to him. Therefore, we added a better distance sensor. Specifically we added the Hokuyo URG-04LX-UG01. It's a good Lidar because it has a respectable range, good angular resolution, "cost-effective", and is kinda lightweight. Using this sensor we could do mapping if we wanted to which means this system is extensible to the broader challeges of the overall navigation problem such as SLAM. Using the Lidar we'll be able to get enough information to do the job we need to.

While it's easy to plug a USB cable into Nao's head, you still have to stick the Lidar somewhere. Nao doesn't have mount points that make it easy to add new hardware. Given that we have a nifty 3D printer (and I know Solidworks) we designed up a little suit of armor for Nao with a big stick coming out of it that we could mount the Lidar to, above his head. This works but the new dynamics destabilize Nao's default gait at certain speeds. This doesn't mess with the navigation algorithm, but to increase the speed this will have to be dealt with, either by changing the rig or... using the arms to

counterbalance the new inertial forces as part of the walking gait.

2.1 Hardware Overview

Ok so, three pieces of hardware here, the Nao, the Lidar, and the mount. Technically, all you need is the Nao since it is a mobile base with distance sensors but the sonars don't do that great so we added the Lidar, and since there's no where to screw it down we designed a mount.

2.1.1 Nao Hardware

Nao is a humanoid by Aldebaran Robotics. 25 DoF, arms, legs, head, hands feet. Sonars, joint sensors, cameras, foot sensors, IMU, bumpers and buttons. Battery life, weight, top speed (before and after Lidar), sonar ranges, camera angles and pixels, CPU type and speed, RAM, storage space, USB, Ethernet, WiFi. Motor torques. (Important for Chapter IV) [Diagram of Nao. More than one to show different things]

2.1.2 Lidar Hardware

Hokuyo is a Lidar. What's a Lidar? How does it work and what does it give you? Diagrams explaining things

Number of beams, angular resolution of beams, range of beams, resolution and accuracy. Weight, power consumption, USB, update rate (sensor bandwidth). Indoor only, not rated for outdoors. [Diagram of Hokuyo.]

2.1.3 Lidar Mount

Design reqs: needed to rigidly attach to Nao for data transformation purposes, needed to see forward, needed to be able to crawl with it, lightweight, Nao needed to be able to move his head while wearing it, needed to be able to use the sonars (just in case).

Mounting it to the head seemed like it was going to be tough, so a vest was designed. Straps and foam seemed like they'd hold well enough. In fact they hold so well I can lift Nao up by the mount. [Show Solidworks design.]

2.2 Software Framework

So Nao comes with a much of good stuff. The API allows us to program in C++ or python and run locally or remote. We can also test things quickly using Choreographe.

Because it's linux we can also easily run other non-aldebaran programs in python or using g++ complier. The lidar runs on a python server, allowing the ground station and the Nao to easily see/use the data since everything is over WiFi and the sockets don't care. How does this work? How to log into Nao. Choreographe, remote run, and ssh for local run.

2.2.1 NaoQi API and qibuild

qibuild is what allows us to use the libraries remotely or locally without thinking. It's based on CMake so you can add new libraries in just as you would with CMake on Linux.

The API is really a lot. You can command things on the joint level, nav level, pose level, set velocities or positions, get orientation, distances, and joint angles or currents. Everything is done with proxies.

2.2.2 User Operation

How do you start Nao, start Lidar? Log in to robot? Start program. Load program. Start algorithms, stop them.

CHAPTER III

Navigation

Navigation is the task of generating a series of commands that allow a robot to transit from its current pose to some goal pose. This task encompasses a number of subtasks such as path planning, environment sensing, and obstacle avoidance. As a robot transits through space from its current to goal pose, it necessarily visits a series of intermediate poses the choice of which depends on its kinematic constraints and the constraints of the environment it transits through. Collectively, this set of poses as a time sequence is known as a path. The spacial region in which the robot operates is known as the task space and all kinematically feasible configurations is known as the configuration space. Typically, the environment a robot traverses is not free space but also contains obstacles which constrain the paths which can be taken to the goal. While in some controlled environments the location of obstacles can be known *a priori* and supplied to the robot in the form of a map, often the robot will have the task of creating a map which describes the location of these obstacles. In some situations, "global" sensors can view the entire environment within which the robot will operate and can capture all of the constraints while in others (especially in the case of mobile robots) only "local" sensors mounted to the robot are available to detect environmental limitations to the robot's path as they are encountered. Even more challenging is the case where the environmental obstacles are not static but are moving adding a time varying element to the obstacle avoidance problem.

Algorithms that solve the navigation problem are sometimes described according to four major categories (though there are many other categories with which these algorithms could be described with). These categories are not mutually exclusive, and algorithms are commonly attributed with multiple classifications. The first two, known as global or local, speak to the amount of spatial data or time horizon considered when planning a solution. If only a short time horizon or occlusions in the immediate spatial region are accounted for, then the algorithm is considered local, otherwise it is thought of

as global. The other two deal with how the spaces or paths are represented, continuously or discretely. If the algorithm breaks up these spaces into finite resolution pieces, then it is discrete. Continuous algorithms choose some sort of parameterization to the spatial or environmental constraints using a model. Often algorithms are some mixture of all of these four categories.

The algorithm used for navigation in this thesis is the GODZILA navigation algorithm, which can be categorized as a local continuous algorithm based on the potential fields navigation strategy. While in many environments GODZILA solves the navigation problem, it can also be used in conjunction with global algorithms that can plan way points to the goal without having to consider the detailed local environment.

In the design on the obstacle avoidance algorithm for a robot, it is important to consider how different local sensors can affect the choice of navigation algorithm. The Nao humanoid platform used in this thesis is equipped with two sonar sensors for obstacle detection. Details on these are given in Chapter II. These sensors have a broad angular range and cannot provide information about where within this cone obstacles are located. This enables the robot to avoid colliding with objects directly in front of it but might cause the obstacle avoidance algorithm to preclude the discovery of possible paths (e.g., narrow corridors between obstacles). Conversely, scanning laser rangefinders (such as the one mounted to the Nao and described in Chapter II) have a very high degree of angular resolution and provide hundreds of range measurements. This allows for a more accurate description of environmental occlusions allowing more paths to be considered for traversal.

The notion of what objects in the environment occlude a path and which do not can be viewed as a function of the gait (or mode of locomotion) used by the robot. If the robot is restricted to a plane, such as the case in many ground vehicles, then an object of similar size to the vehicle could prevent traversal through that position. If the robot can also fly then that object may not present an impediment to proceeding towards the goal. While still not transiting through that exact position in 3D space, the 2D projection of the path onto the plane will appear to have moved through the obstacle. This can be though of in terms of a transformation of the obstacle set according to the different motion modalities, making a 2D planner still applicable to the problem. In Chapter IV, a crawling gait for the Nao robot is considered which allows it to go under occlusions that it would otherwise need to plan around.

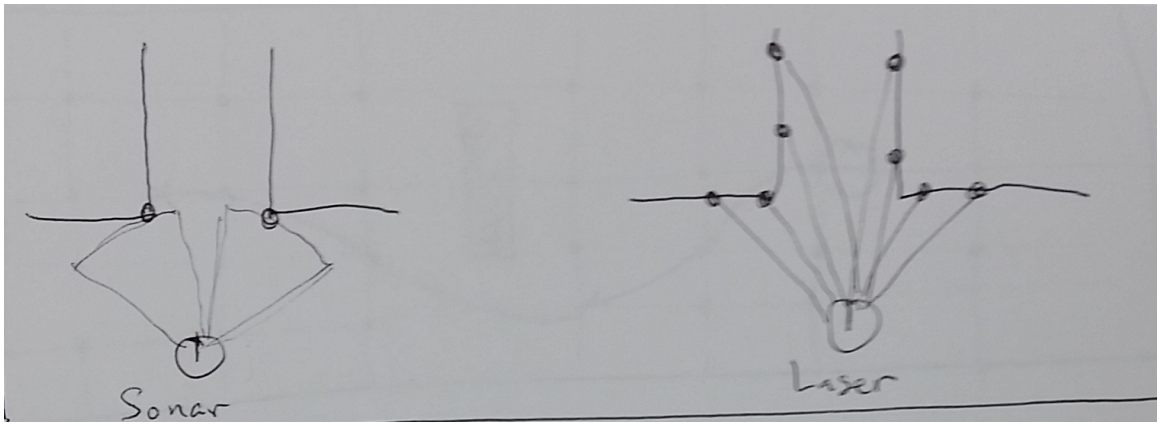

Figure 3: Sonar sensors are typically preferred when needing to detect the presence of an obstacle without needing precise position information. The location of a chair leg in the path of a robot would be detected by sonar while not knowing its position in the sonar cone. Laser rangefinders have very narrow beam width, typically requiring many of them to be effective at describing an environment. If many measurements are available at high angular resolution, then they can be very effective at describing the width of narrow apertures.

3.1 Algorithm Classifications

Here some of the different notions common to navigation algorithms are explored. These classifications are sometimes useful for the navigation engineer to think about as tools available to solve the navigation problem. While additional classifications and subclassifications of algorithms can be considered, the following description is intended as a brief overview of the essential concepts.

3.1.1 Global

One approach to the navigation problem is to consider the entire task and configuration space of a problem when generating a solution. For example, the task space of a robot arm might be the pose of the end effector and all of the occlusions within that space. The configuration space would be the set of all valid joint angles that the arm can attain. Using this, an algorithm can compute a time sequence of poses or movement commands that the robot should execute in order to bring it from its current pose to the final pose. While this approach is ideal in the sense of generating a solution to the original problem, it has several practical disadvantages. One requirement of the algorithm is having a complete description of the task space. In many cases

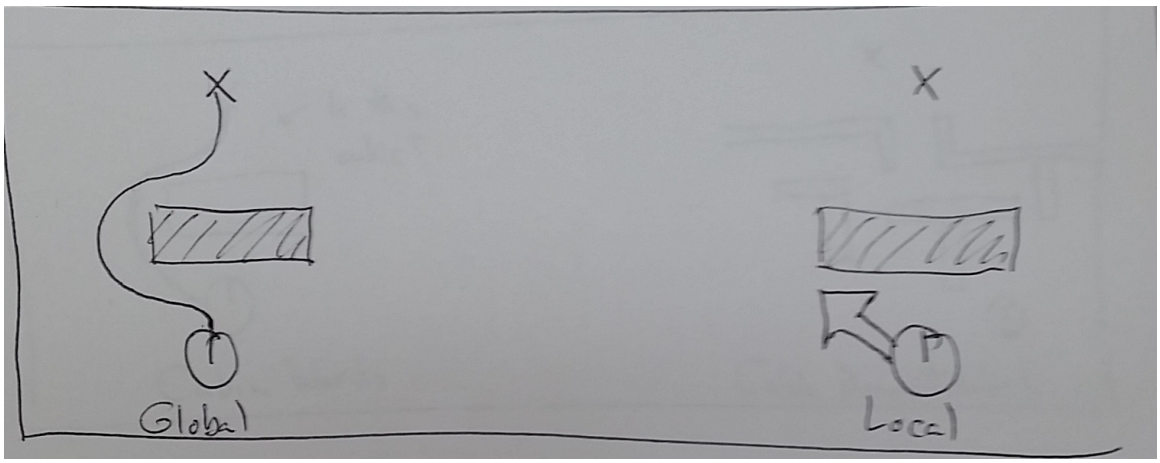

Figure 4: In global approaches, an entire path is planned to the goal before commanding the robot. In local schemes, the robot is given commands calculated as a function of its position and the local environment and a complete path to the goal is typically precomputed.

global knowledge of occlusions while planning is not available meaning that when new obstructions are observed, the algorithm needs to replan the path. The other commonly encountered problem with these algorithms is when the magnitude of the space to be planned through is so large that computing a solution cannot be done in real-time.

3.1.2 Local

A different approach to the navigation problem is generating commands based only on the current information available or a short history about the environment. In the robot arm example, there might be a sensor such as a camera that can detect objects that are obstructing the end effector from reaching the goal and instructs the arm to move around it. Such algorithms usually have the advantage of being quick to compute as compared to their global counterparts because they only have to consider a small subset of the task and configuration space. Their main disadvantage is that since they only use such a limited amount of information, they can become trapped in local optima that do not allow the robot to achieve the desired pose.

3.1.3 Discrete

Orthogonal to the ideas of local and global path planning which deal with the scope of the time and space being considered when generating navigation commands is how these spaces are represented. One way to represent the space is to break it up into a set

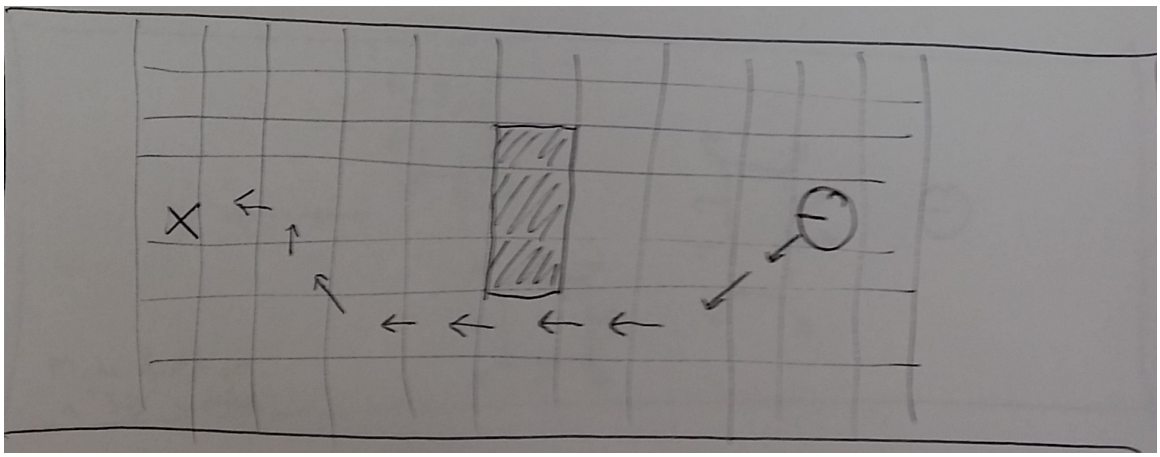

Figure 5: The environment here is being represented as occlusions in a discrete grid of locations. This allows the robot to plan through successive locations to the goal using graph-based techniques.

of discrete locations and then plan through that discretized representation. The task space for example could be divided into uniform spatial regions which an algorithm can consider visiting when planning a path. Alternatively, as with the robot arm example, a finite set of movement commands can be considered and iterated through while generating a solution. Graph-based search algorithms such as Dijkstra's algorithm or A* are examples of discrete solvers. An advantage to this is that paths with complex shapes can be generated to accommodate difficult constraints. One point however to consider in this approach is how finely to resolve the task or configuration spaces. Coarser discretization can allow a solution to be computed rapidly but might miss more optimal solutions.

3.1.4 Continuous

Continuous spatial representation avoids the problem of having to choose a discretization resolution. The movement commands or paths are planned in a continuum allowing paths to take on intermediate values not available at a given discrete resolution. Continuous representation instead has a different problem of paths needing to take on particular solution forms. A path through the task space might be represented as a cubic spline from the current pose to the goal pose. This path has access to the entire task space but may not be able to construct a path under complicated environments with many obstacles. Often instead of using a single cubic, designers will use a series of cubics, the termination of one being the starting position of the next, until the goal

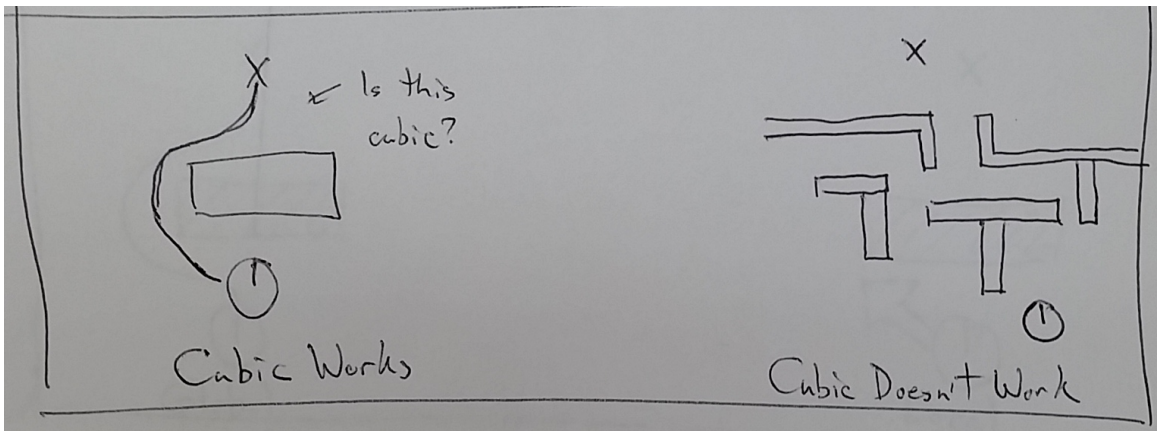

Figure 6: This figure shows two sample environments. On the left, the robot can follow a trajectory that takes a cubic form and reaches the goal location. On the right, a more complicated environment is shown where a single cubic cannot describe an appropriate path.

is reached. Alternatively, instead of restricting the path representation to a series of cubics, other trajectory forms can be utilized as a library of representations such as higher-order polynomials, trigonometric functions, etc. One example of such a planner is known as a "maneuver-based" planner.

3.1.5 Composite Approaches

In order to combine the strengths and combat the weaknesses, the above approaches are often combined to form a more robust navigation solution. Global plans can be generated as a series of intermediate goals or way points for local planners to navigate towards. These way points can be planned on a coarse discrete grid that is quick to compute while low dimensional continuous trajectories are plotted between them. Lattice planners are an example of such a composite algorithm.

3.2 Potential Fields

Potential fields algorithms are continuous local planners. They work on the concept of modeling the robot as a sort of charged particle (like an electron) and obstacles in the environment are modeled as having the same polarity of charge as the robot thereby producing a repulsive field pushing the robot away from them. The goal location in is modeled as having an opposite charge to the robot and thus produces an attractive

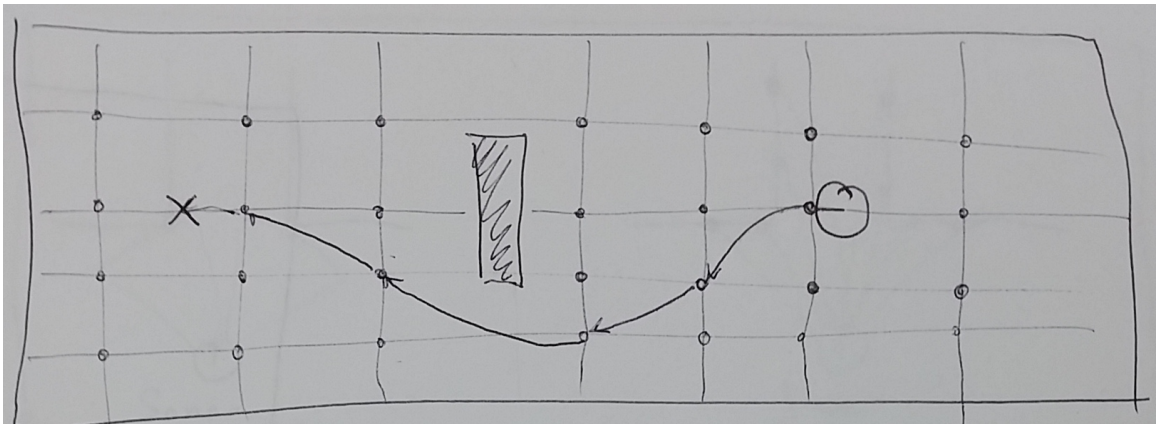

Figure 7: A lattice planner is an example of a composite discrete and continuous path planner. The planner first plans through a discrete grid and then designs trajectories that bring the robot from each point in the planned discrete grid to the next.

force, pulling the robot towards it. As the robot moves through the environment, it detects objects and generates motion commands based on its range and direction.

Given a vector x_r which describes the position of the robot in the task space and the positions of obstacles m_i in the set of all obstacles M, a force vector f_i can be generated for each obstacle that describes the magnitude and direction of the repulsive force applied to the robot by the obstacle. The force direction is in the opposite direction of the relative bearing of the obstacle to the robot with magnitude being inversely proportional to the distance of the obstacle from the robot. Equation (3.2) shows an example of force vector generation.

$$\|f_i\| = \frac{-c_i}{\|m_i - x_r\|^\alpha} \tag{3.1}$$

$$\angle f_i = \angle(m_i - x_r) \tag{3.2}$$

$m_i - x_r$ is the location of the obstacle relative to the robot and c_i is a positive scalar used to tune force magnitude. In some implementations, c_i can be a function of the relative bearing, for example, decreasing the repulsion effect if the obstacle is not in the direction of the goal. The parameter α in general can be any positive scalar but is often picked to be 2.

For the goal seeking behavior, the equations are similar, with the direction of the force being in the direction of the goal, rather than away from it.

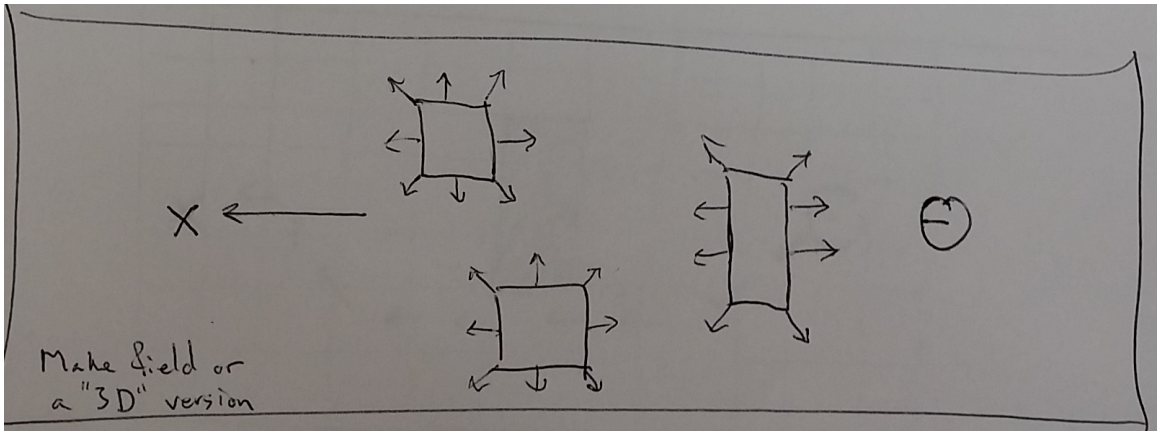

Figure 8: Visualization of potential fields idea. Objects in the environment emit a field that repels the robot and the goal emits an attractive field. This can be visualized as a 2D or 3D vector field.

$$\|f_g\| = \frac{c_g}{\|x_g - x_r\|^\gamma} \tag{3.3}$$

$$\angle f_g = \angle(x_g - x_r) \tag{3.4}$$

$x_g - x_r$ in equation (3.4) is the relative location of the goal with respect to the robot and c_g is some positive tuning scalar. γ is some positive scalar, typically 2.

The sum of these forces is used to produce a force vector f_r used to command the robot away from obstacles and towards the goal location.

$$f_r = f_g + \sum_{i=1}^{\|M\|} f_i \tag{3.5}$$

where $\|M\|$ is the number of obstacles (modeled as a discrete set).

3.3 GODZILA

GODZILA (Game-theoretic Optimal Deformable Zone with Inertia and Local Approach) [1] is a local continuous navigation algorithm based on the potential fields idea. In contrast to some potential fields formulations which work on the range and bearing of objects in the environment, GODZILA uses the sensor information about occlusion locations directly in order to formulate navigation commands. It is a memoryless algorithm and does not attempt to build a map of the environment making it very lightweight in

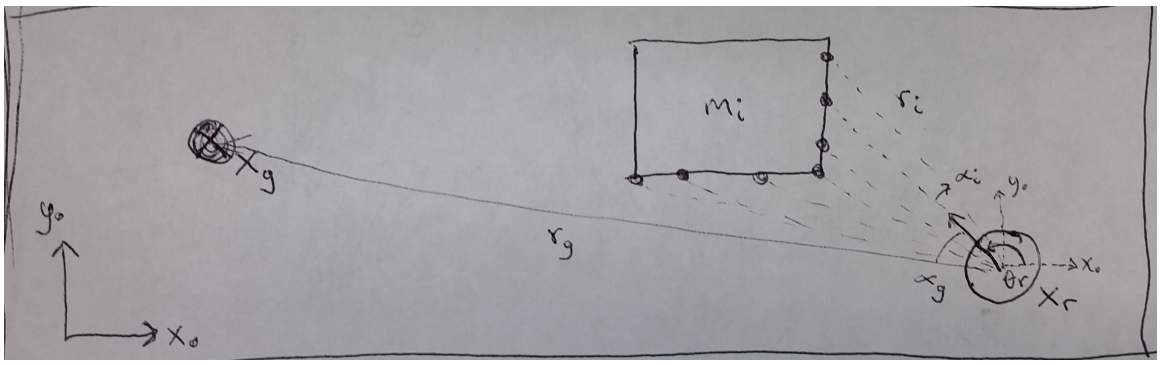

Figure 9: Illustration of various variables utilized in GODZILA. Range and bearing of the goal relative to the robot are required as well as being able to take range measurements to occlusions and knowing the relative bearings of these measurements.

terms of both computation and memory. It can be implemented on a variety of vehicles and is suited for the cases where a low computational power microcontroller is utilized. It has three components which are an optimization cost, a straight line planner, and a stochastic local minima escape strategy.

Figure 9 illustrates the key variables in the formulation.

Taking a 2D example of the application of the GODZILA algorithm, we have the following terms:

x_0 is the x-axis of the inertial frame.

y_0 is the y-axis of the inertial frame.

x_r is the (x,y) location of the robot in the inertial frame.

α_r' is the new heading the robot should be commanded to in the robot frame.

x_g is the (x,y) location of the goal in the inertial frame.

r_g is the range to the goal relative to the position of the robot.

α_g is the bearing to the goal location relative to the orientation of the robot.

z_i is the i^{th} sensor measurement made by the robot which includes range and bearing to an occlusion.

r_i is the i^{th} range measurement to an occlusion relative to the position of the robot.

α_i is the bearing angle to the i^{th} sensor measurement.

In this formulation, it is assumed that there is some mechanism to allow the robot to get an estimate of the relative range and bearing to the goal location r_g and α_g and that the robot has some sensors that can provide range and bearing to occlusions in the environment.

3.3.1 Optimization Formulation

The formulation of the GODZILA algorithm is in terms of an optimization cost. The goal of the algorithm is to bring the robot towards the goal while preventing it from colliding with obstacles. With this intuition the cost function has three components:

1. $J_1(\alpha'_r)$ which rewards goal seeking by penalizing directions other than those towards the goal.
2. $J_2(\alpha'_r)$ which penalizes directions towards occlusions.
3. $J_3(\alpha'_r)$ which dampens oscillations by penalizing changes in heading.

The sum of these terms produces the cost function to be minimized:

$$\min_{\alpha'_r} \sum_{i=1}^{3} J_i(\alpha'_r) \tag{3.6}$$

where α'_r is the heading direction the robot should travel in order to minimize the cost function. $J_1(\alpha'_r)$ takes on the form of:

$$J_1(\alpha'_r) = g_{11}(r_g) f_{11}(\|\alpha'_r - \alpha_g\|). \tag{3.7}$$

g_{11} is a class-$\mathcal{L}$ function meaning that it is continuous and monotonically non-increasing which maps $[0, \infty) \mapsto [0, \infty)$ and f_{11} is a class-$\mathcal{K}$ function meaning it is monotonically non-decreasing which maps $[0, \infty) \mapsto [0, \infty)$. The intuition here is that f_{11} produces a higher cost in proportion to how much the direction command α'_r differs from the goal direction. g_{11} conversely reduces the cost if the range to the goal is large. The idea being that the farther from the goal the robot is, then being oriented towards the goal becomes less of a priority.

$J_2(\alpha'_r)$ takes on the form:

$$J_2(\alpha'_r) = J_{2_{\mathcal{I}_1}}(\alpha'_r) - J_{2_{\mathcal{I}_2}}(\alpha'_r) \tag{3.8}$$

$$J_{2_{\mathcal{I}_1}}(\alpha'_r) = \sum_{i \in \mathcal{I}_1} g_{21}(r_i) \Big[g_{22}(\|\alpha_g - \alpha_i\|) + g_{23}(\|\alpha_i\|) \Big] g_{24}(\|\alpha'_r - \alpha_i\|) \tag{3.9}$$

$$J_{2_{\mathcal{I}_2}}(\alpha'_r) = \sum_{i \in \mathcal{I}_2} f_{21}(r_i) g_{25}(\|\alpha_g - \alpha_i\|) g_{26}(\|\alpha'_r - \alpha_i\|) \tag{3.10}$$

Equation (3.8) has two components to it, separated by the membership of the sensor measurements z_i to one of two sets. The first is the set of all range measurements r_i whose value falls below some positive scalar r_c are put into a set $\mathcal{I}_1$. The second set is the remaining range measurements, which are necessarily greater than r_c, placed into set $\mathcal{I}_2$. This threshold distance r_c is picked such that if the occlusion is farther away than it, the robot does not need to be concerned with avoiding it. Equation (3.9) is concerned with the occlusions that need to be avoided, and has four class-$\mathcal{L}$ functions $g_{21}, g_{22}, g_{23}, g_{24}$ associated with it. The first three g_{21}, g_{22}, g_{23} act as gains that the control variable α'_r in function g_{24} must attenuate. g_{21} says that if the range r_i to the occlusion is small, then the cost of orienting in that direction is high. g_{22} amplifies this effect if the direction to the occlusion α_i is in the same direction of the goal α_g. g_{23} also amplifies g_{21} if the direction of the occlusion is similar to the heading direction of the robot. Equation (3.10) deals with the case where the occlusions can be approached. It has one class-$\mathcal{K}$ function f_{21} and two class-$\mathcal{L}$ functions g_{25} and g_{26}. f_{21} and g_{25} act as gains on the controlled function g_{26}. f_{21} promotes moving in the direction of ranges that are large and g_{25} promotes moving in the direction of occlusions that are in a similar direction to the goal direction.

The final term $J_3(\alpha'_r)$ is a class-$\mathcal{K}$ function that penalizes changes in direction.

$$J_3(\alpha'_r) = f_{31}(\|\alpha'_r\|) \tag{3.11}$$

This term is present to prevent high frequency oscillations and is analogous to giving the robot a physical inertia. All of the functions $f_{11}, f_{21}, f_{31}, g_{11}, g_{21}, g_{22}, g_{23}, g_{24}, g_{25}, g_{26}$ in the optimization are tunable by the designer but it was noted in [1] that in the case where the controlled functions $f_{11}, g_{24}, g_{26}, f_{31}$ are chosen to be quadratic, the optimization problem can be solved in closed form producing the solution seen in equation (3.12).

$$\alpha_r' = \frac{\alpha}{\|\alpha\|} \tag{3.12}$$

$$\alpha = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 \tag{3.13}$$

The four terms in equation (3.13) correspond to terms arising from the four optimization terms $J_1(\alpha_r')$, $J_{2_{\mathcal{I}_1}}(\alpha_r')$, $J_{2_{\mathcal{I}_2}}(\alpha_r')$, $J_3(\alpha_r')$. They are expanded in equations (3.14)-(3.17).

$$\alpha_1 = \overline{f}_{11} g_{11}(r_g)\alpha_g \tag{3.14}$$

$$\alpha_2 = -\overline{g}_{24} \sum_{i \in \mathcal{I}_1} g_{21}(r_i)\Big[g_{22}(\|\alpha_g - \alpha_i\|) + g_{23}(\|\alpha_i\|)\Big]\alpha_i \tag{3.15}$$

$$\alpha_3 = \overline{g}_{26} \sum_{i \in \mathcal{I}_2} f_{21}(r_i) g_{25}(\|\alpha_g - \alpha_i\|)\alpha_i \tag{3.16}$$

$$\alpha_4 = \overline{f}_{31}\alpha_r \tag{3.17}$$

The solutions to the equations reuse functions $f_{21}, g_{11}, g_{21}, g_{22}, g_{23}, g_{25}$ as they are not functions of α_r' and produce constant versions of $f_{11}, f_{31}, g_{24}, g_{26}$ as $\overline{f}_{11}, \overline{f}_{31}, \overline{g}_{24}, \overline{g}_{26}$ in the direction of their respective directions $\alpha_g, \alpha_i \forall i \in \mathcal{I}_1, \alpha_i \forall i \in \mathcal{I}_2, \alpha_r$ where $\alpha_r = [1, 0]^T$ representing the current heading of the robot in the vehicle frame.

The resultant α_r' is typically commanded as an angular rate $\dot{\alpha}_r'$ according to the angular velocity bandwidth of the vehicle.

Finally, the linear velocity is still to be commanded. A good choice for this function is of the form:

$$v_r = f_v(\min(R))g_v(\dot{\alpha}_r') \tag{3.18}$$

where R is the set of all range measurements, f_v is a class-$\mathcal{K}$ function and g_v is a class-$\mathcal{L}$ function. This reduces the speed of the vehicle when it is in close proximity to occlusions or if it is commanded to a high angular rate. Slower speeds near obstacles makes it less likely that the vehicle will collide with them. Slower speeds at times when the robot is commanded to high angular rates helps reduce the distance the robot travels in a previously commanded direction before completing a new turn command.

3.3.2 Straight Line Planner

If at some point during the navigation of the vehicle to the goal the straight line path from the robot to goal becomes unobstructed, it follows that the robot should proceed

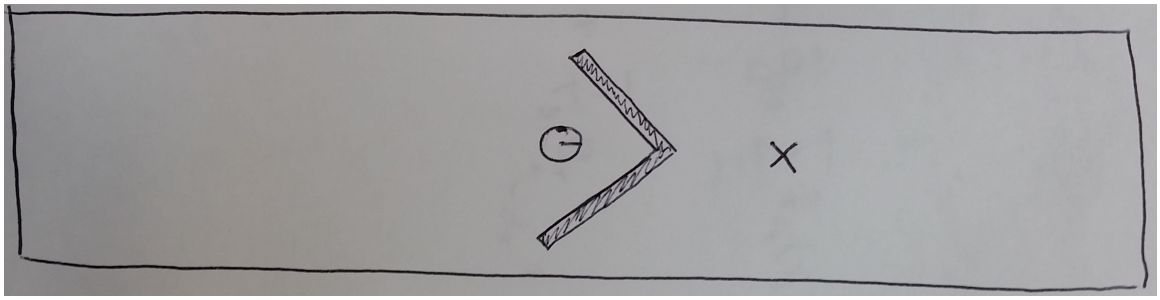

Figure 10: A degenerate case where the attractive force of the goal combines with the repulsive force of the wall producing a local optima where the robot gets trapped.

directly to the goal according to this path. This is sensible, but can bring the robot in close proximity to obstacles. It is therefore desirable to preserve the obstacle avoidance properties of the optimization procedure from 3.3.1. This procedure is especially useful in situations where the width of the aperture between occlusions to the goal approaches the width of the robot where the obstacle avoidance behavior might deter traversal to the goal. This can be achieved by prescribing intermediate goal locations according to:

$$\hat{x}_g(t) = (1 - \lambda(t - t_0))x_r + \lambda(t - t_0)x_g \tag{3.19}$$

where $\lambda(\tau)$ is a monotonically increasing function that maps $[0, T_f] \mapsto [0, 1]$ with T_f being some amount of time allowed for the vehicle to reach the goal location. The length of time the planner is active is T_f. If the robot has not reached the goal in this time but again has a straight line path to the goal the planner is allowed to restart.

3.3.3 Escape Strategy

While the inertial term provided by (3.11) is intended to reduce high frequency oscillations and backtracking that might trap the robot in small corners, it is not enough to account for degenerate cases such as those shown in Figure 10.

In such scenarios, an effective strategy to escape such traps is to randomly assign an alternate goal position x_{rand} that will allow the robot to escape the local minima and proceed to the goal. The distance to x_{rand} from the robot is typically selected to be smaller than the distance to the nearest occlusion $\min(R)$. After some fixed time, the goal position is set back to x_g. If the trap condition is detected multiple times then the next escape procedure is repeated that many times before being restored to x_g. One possible method to detect a trap condition is to integrate a sliding window of vehicle linear velocities v_r and if those values integrate to a position close to zero then is is

likely that the robot has encountered a trap condition. It is shown in [1] that with all of these mechanisms the position of the robot x_r will converge to x_g in finite time with probability 1.

CHAPTER IV

Humanoid Crawl Gait

Many robots that enjoy practical use are not mobile. They are affixed to a table or floor and do not depend on their environment in order to produce a commanded movement. They utilize a motion model that describes how their parts interact with each other to move an end effector and typically avoid interacting with objects in their environment other than those they are required to manipulate. Mobile robots also use a motion model to plan their movements but, in contrast to fixed robots, must interact with their environment to produce movement. While this can be a more challenging problem than motion planning for fixed robots, the use of the environment to produce motion leads to mobile robots having a richer and more adaptable set of movements. Aerial and aquatic vehicles push against water or air, wheeled robots roll over the ground, legged robots walk, run, and crawl. Not only can mobile robots operate through a wider set of environments than fixed robots, but different actuation schemes can be employed to produce different movements. Legged robots are are a particularly interesting platform for ground locomotion, as opposed to wheeled platforms, because they are more adaptable to a wide range of environments by actuating their legs according to a different strategy. Collectively, the method by which legged robots actuate their legs as a function of time is called gaiting. Different gaits produce different characteristics such as the range of achievable speeds, endurance, terrain adaptability and a host of other things. As was described in Chapter III, the gaiting modality directly affects the navigation strategy. By selecting the appropriate gaiting strategy, the robot can modify its movement to achieve a commanded goal. This modified movement also modifies which environmental objects present as obstacles. Expanding the library of gaits legged robots have access to allows robots to be increasingly more capable and applicable to a wider range of scenarios.

Using the Nao platform, crawling gaits applicable to humanoid robots was explored. Not only is crawling a more stable gaiting strategy than walking, but it gives the robot

access to areas of the environment that are inaccessible via walking alone. A low-profile laterally symmetric crawl gait is described, modeling the robot as a closed-chain manipulator with pseudo-static dynamics. This gait was parameterized on three joint variables and optimized using a cubic splines via a genetic algorithm.

4.1 Humanoid Crawling

Unlike walking and running which enjoy precise definitions, crawling seems to only have a subjective notion. [2] asserts that crawling is a statically stable walk. This definition is problematic as bipedal gaits have been demonstrated in [**?**] that are statically stable but would not be classified as walking. In addition to this, soldiers performing the high army crawl [3] can be seen to perform this motion very quickly which introduces a dynamic component to the crawl. The standard or baby crawl is described as using one's hands and knees to produce forward motion. In contrast to this, crawls such as the leopard, tiger, bear, and crab use hands and feet to produce forward motion and the low army crawl uses the hands to drag and one leg to push the body across the ground. Such diversity in crawling motion makes it difficult to differentiate a crawl from a statically stable quadrupedal gait that uses something other than the end effectors to interact with the environment. Despite this, the presented gait produces a motion that many would associate with humanoid crawling.

4.1.1 Nao Crawling Limitations

A primary limitation to the gaits that can be produced by the Nao is the limited number of degrees of freedom (DoF) of the platform in contrast to the large number of DoF present in humans. While the Nao has 25 DoF, the human body has 244 [4]. The human arm and leg each have 7 DoF while the Nao's arms and legs each have 5. Nao's hips have one more degree of freedom, called Hip Yaw-Pitch, which turn the legs together at an angle. Figure 11 illustrates this degree of freedom more clearly. The rest of the DoF of the Nao are in the hands and neck. Notably, Nao has no back joint. This prevents the gait designer from prescribing a twisting motion for use in the crawl gait. These limits in motion preclude the execution of any gaits that require lateral twisting or sagittal arching. Examples of this are shown in Figure 12

Figure 11: The pane on the left shows the leg configuration of the Nao when the Hip Yaw-Pitch DoF is fully turned in. The right pane show the leg configuration when the Hip Yaw-Pitch is fully turned out. The legs are mechanically linked, making the amount of Hip Yaw-Pitch equal for each leg.

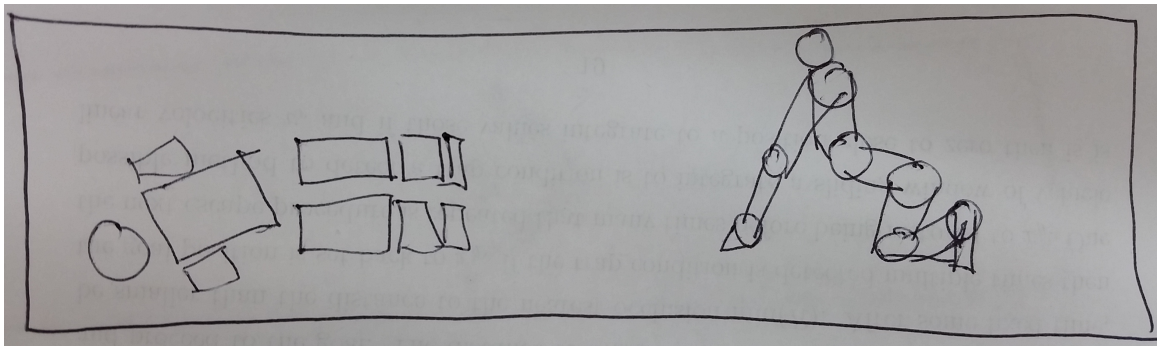

Figure 12: Illustration of crawl motions that involve actuated back joints. The left pane shows lateral twisting during a crawl. The right panes shows sagittal arching while on hands and knees.

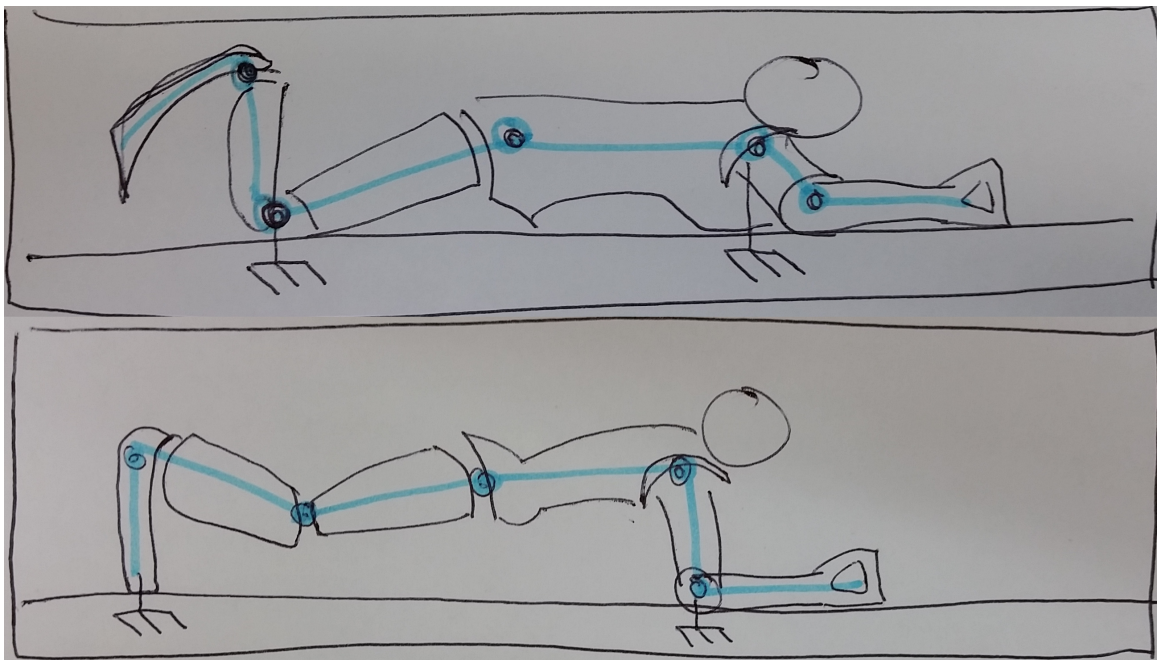

Figure 13: Illustration of Projected Profile concept. Both panes show the saggital view of the Nao with a schematic representation of the kinematics. In the top view, the robot appears as two open chain manipulators. In the bottom view, the robot appears as one closed chain manipulator.

4.2 Projected Profile Humanoid Crawl Gait

The crawling gait presented in this thesis is based on viewing the humanoid form as a set of manipulators on the sagittal plane. Figure 13 illustrates this concept. When the robot is laying on its underside, it necessarily makes contact with the ground. If we then view the robot from the side (looking at the sagittal plane) we can model it as a planar kinematic chain. If the chest and knees are making contact with the ground, then the arms from the shoulder joint to the hand, and the legs from the knees to the toes are free to move without effecting the rest of the body. This produces two open chain manipulators. In the case of the Nao, each has two degrees of freedom in the sagittal pane, allowing the hands and feet to move independently.

If the elbows and toes are placed on the ground, then the body from the toes to the elbow can be viewed as a closed chain manipulator. This allows the joints to work together to move the center of mass. Kinematically, these two phases share two common configurations. The first configuration is when the elbow is at full extension and the toes touching the knees. We will call this the “extension” configuration. This can be

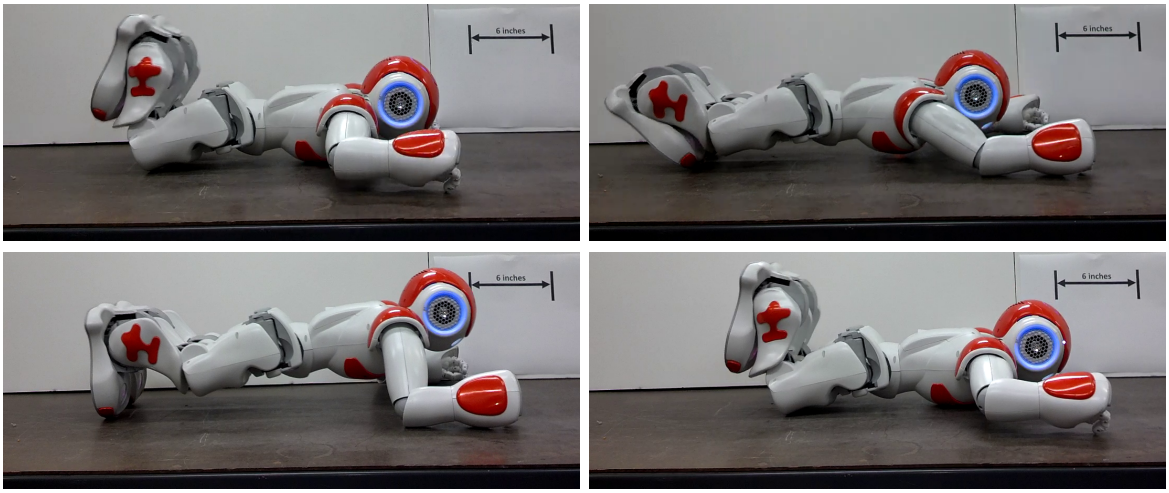

Figure 14: The above sequence shows the motion segments and robot postures in the crawl gait. The upper left pane shows the initial open chain configuration. The upper right pane moves the robot to the "extension" configuration. The lower left shows the "compression" configuration. Finally, the lower right shows the robot, having translated forward, once again in the open chain configuration. A 6-inch marker is shown in the background as a length scale reference.

viewed as the robot reaching forward. The second is when the elbow is at full flexion and the ankles at extension. We will call this "compression". This can be viewed as the robot having pulled itself forward. These motions can then be combined to produce a full gait.

Using the Nao robot as an example, Figure 14 shows the full gaiting sequence. The robot initializes itself in the open chain configuration. From this is can position its end effectors (the toes and elbows) into the first common configuration "extension". Next, the robot is in the closed chain configuration in which it can transport its center of mass forward until it reaches the "compression" configuration. Finally the robot is again in the open chain configuration and the cycle can start again.

The gait is laterally symmetric. If we actuate the joints at an appropriate rate, dynamic effects from the robots motion do not become a significant factor. As detailed in Chapter **??**, this gait can be performed on the Nao as a speed of 1 ft every 6 to 8 seconds. The wide surface area of the forearm proves a high coefficient of friction against slipping and the small surface area of the toes can act as a point of high pressure which can dig into soft surfaces such as carpets. The gait is static in the sense that the robot's motion can be paused at any point and the robot will not fall. The gait does not depend

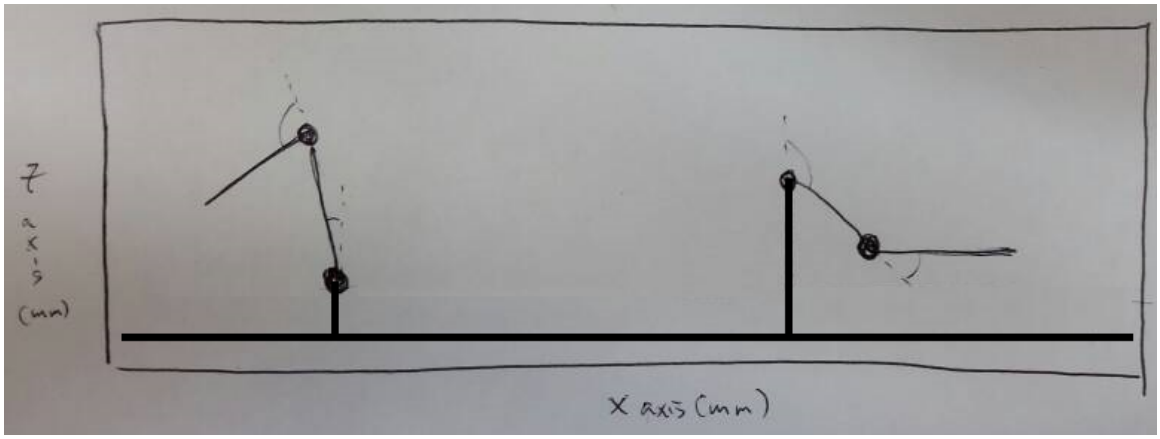

Figure 15: Simplified kinematic model of the sagittal projection of the open chain configuration. The manipulator on the left represents the tibia-foot chain. The manipulator on the right represents the humerus-forearm chain. The origin of the knee and shoulder are represented as having a z-axis offset because the knee and chest of the robot have a height that raises their origins.

on the robot sliding along the surface nor does it depend on friction to pull the robot forward. The gait has a very low profile. The highest point on the robot during the gait (which is the top of the head) is about 8 inches off of the ground. In contrast, during walking the Nao robot stands 23 inches tall.

4.2.1 Open Chain Kinematics

The Projected Profile crawl gait has two kinematic configurations: open chain and closed chain. In the open chain configuration the robot acts as two independent planar manipulators. Each manipulator has two degrees of freedom as can be seen in Figure 15

With the Nao facing downwards, the feet towards the origin and the head in the positive x direction, the forward kinematics for each manipulator are described by:

$$x = x_0 + l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) \tag{4.1}$$

$$z = z_0 + l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) \tag{4.2}$$

where x_0 and z_0 are the superior and posterior offsets with respect to the global frame, respectively. l_1 is the length of the humerus/tibia, l_2 is the length of the forearm/foot, θ_1 is the angle made by the shoulder/knee with respect to the x-axis, and θ_2 is the angle made by the ankle/elbow with respect to the x-axis of the humerus/tibia.

The solutions to the inverse kinematics problem can be seen as:

$$\theta_2 = cos^{-1}\left(\frac{(x-x_0)^2 + (z-z_0)^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \tag{4.3}$$

$$\theta_1 = 2tan^{-1}\left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}\right) \tag{4.4}$$

$$A = (x-x_0) + (z-z_0) + l_1 + l_2(cos(\theta_2) + sin(\theta_2)) \tag{4.5}$$

$$B = -2(l_1 + l_2(cos(\theta_2) - sin(\theta_2)) \tag{4.6}$$

$$C = (x-x_0) + (z-z_0) - l_1 - l_2(cos(\theta_2) + sin(\theta_2)) \tag{4.7}$$

This solution derives from the standard inverse kinematics procedure of inverting the forward kinematics. θ_1 must be chosen such that no part of the robot tries to intersect with the floor. In practice, $atan2$ is used in the place of tan^{-1} .

4.2.2 Closed Chain Kinematics

The closed chain configuration models the toes and elbows of the robot as being fixed to the ground. As with all closed chain kinematics, describing the forward kinematics requires solving an inverse kinematics equation. Modeling the robot in the same orientation as the open chain, with the toe at the origin and neglecting the thickness of the elbow, the forward kinematics of the closed chain are:

$$d_e = \sum_{i=1}^{5} l_i \cos(\sum_{j=1}^{i} \theta_j) \tag{4.8}$$

$$0 = \sum_{i=1}^{5} l_i \sin(\sum_{j=1}^{i} \theta_j) \tag{4.9}$$

$$\alpha = \sum_{i=1}^{5} \theta_i \tag{4.10}$$

where d_e is the prescribed distance of the elbow from the foot and α is the desired angle created by the x-axis of the humerus and the ground. θ_1 through θ_5 are the angles of the following joints with respect to their previous x-axis: toe-to-ground, ankle, knee, hip, shoulder. In Figure 16 the red lines represent the x-axes. These equations are the standard planar manipulation equations, treating the foot as the base link with the elbow as the end effector. Equation (4.8) constrains the end effector to be a set distance

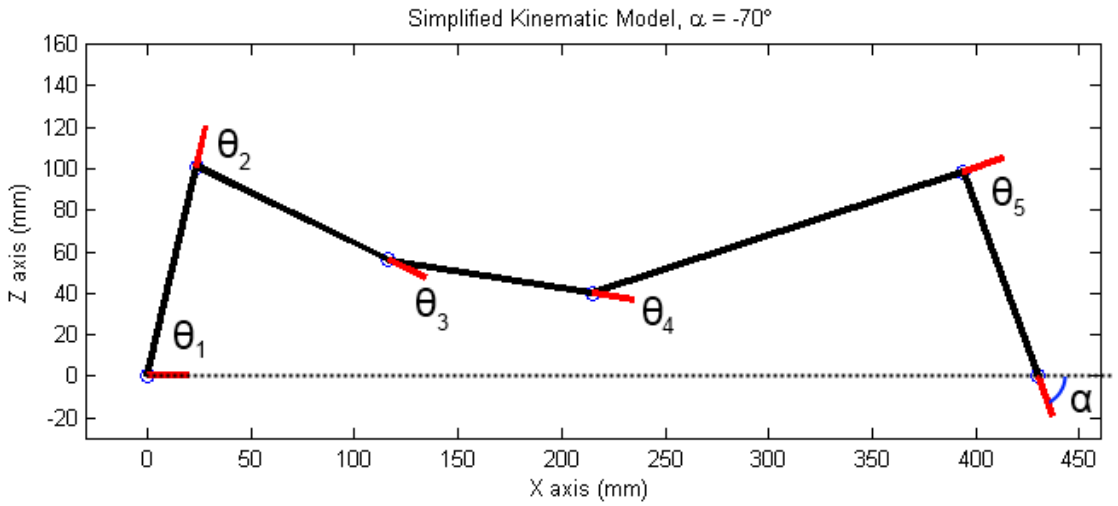

Figure 16: Simplified kinematic model of the sagittal projection of the closed chain configuration.

from the foot and equation (4.9) constrains the end effector to be on the ground. Using equation (4.10), equations (4.8) and (4.9) can be rewritten as:

$$\sum_{i=1}^{4} l_i \cos(\sum_{j=1}^{i} \theta_j) = d_e - l_5 \cos(\alpha) \tag{4.11}$$

$$\sum_{i=1}^{4} l_i \sin(\sum_{j=1}^{i} \theta_j) = -l_5 \sin(\alpha). \tag{4.12}$$

If θ_3, θ_4, and α are prescribed angles, then equations (4.11) and (4.12) are two equations in the two unknowns θ_1 and θ_2. θ_3 and θ_4 can be constant or time-varying as the resultant configuration is a function of these two "free" variables. Taking the square of each of the equations (4.11) and (4.12), an equation in the single variable θ_2 is obtained as:

$$2l_1K_1\cos(\theta_2) + 2l_1K_2\sin(\theta_2) = [d_e - l_5\cos(\alpha)]^2 + [l_5\sin(\alpha)]^2 - l_1^2 - K_1^2 - K_2^2 \tag{4.13}$$

$$K_1 = l_2 + l_3\cos(\theta_3) + l_4\cos(\theta_3 + \theta_4) \tag{4.14}$$

$$K_2 = -l_3\sin(\theta_3) - l_4\sin(\theta_3 + \theta_4) \tag{4.15}$$

The solution to equation (4.13) is of similar form as that seen in (4.4). In general, there will be two solutions for θ_2, but one of them will cause the robot to collide with

the ground and must be guarded against. With θ_2 solved, equations (4.11) and (4.12) can be used to solve for θ_1:

$$\theta_1 = tan^{-1}\left(\frac{sin(\theta_1)}{cos(\theta_1)}\right) \tag{4.16}$$

$$cos(\theta_1) = \frac{K_3(d_e - l_5\cos(\alpha)) + l_5 K_4 \sin(\alpha)}{K_3^2 + K_4^2} \tag{4.17}$$

$$sin(\theta_1) = \frac{K_4(d_e - l_5\cos(\alpha)) - l_5 K_3 \sin(\alpha)}{K_3^2 + K_4^2} \tag{4.18}$$

$$K_3 = l_1 + K_1 cos(\theta_2) + K_2 sin(\theta_2) \tag{4.19}$$

$$K_4 = K_2 cos(\theta_2) - K_1 sin(\theta_2) \tag{4.20}$$

Lastly, θ_5 is solved using equation (4.10).

With these angles solved, the entire robot is parameterized on three angles $\theta_3, \theta_4, \alpha$. If θ_3 and θ_4 are fixed, then starting with the elbow at full extension, bringing the elbow to flexion moves the robot forward. This corresponds to α starting with a small negative angle and ending it with a large negative angle. For the Nao, α is initialized at approximately $-30°$ and terminates at about $-90°$. The primary intuition about this procedure is that the closed chain is like a parallelogram that is used to shift the mass of the robot. Any robot (humanoid or not) that can be set into this configuration can use this framework in order to gait the robot.

4.2.3 Application onto Nao

Once the projected profile time sequence of angles has been computed, it needs to be applied to the Nao. Figure 17 illustrates the sagittal view of the robot in the closed chain configuration. When Nao is set to this configuration, the ankle pitch, knee pitch, hip pitch, and shoulder pitch joints of the robot directly correspond to θ_2 through θ_5. θ_1 corresponds to the angle subtended by the robot's foot and the ground. Unlike the first five joint angles, angle α does not have a direct correspondence and must be derived from the projection of the arm onto the sagittal plane n_p .

[More to come on this part. Continue reading.]

4.3 Optimization

In the previous section, the Projected Profile crawl gait was parameterized on angle triplet $[\theta_3, \theta_4, \alpha]$. To achieve a crawling gait, θ_3 and θ_4 can be set to be constant and α linearly incremented from an initial angle to a final angle as a function of time. While this

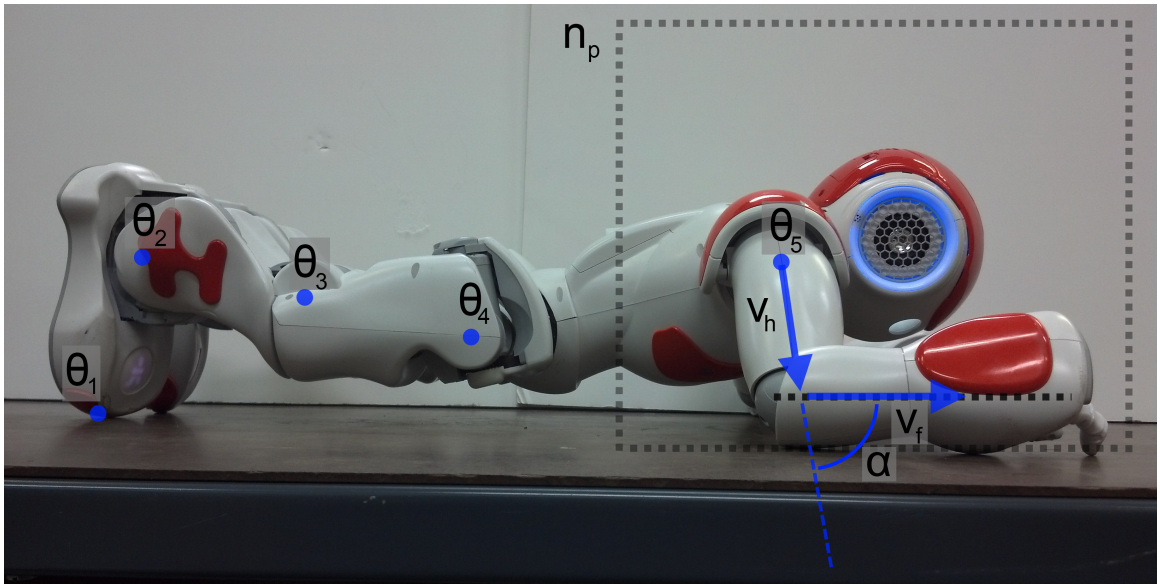

Figure 17: Sagittal view the Nao in the closed chain crawling configuration. The locations of the joints used in the projected profile calculation are shown as blue dots. n_p represents the sagittal plane of the robot.

configuration successfully gaits the robot, it is heuristic. To improve the approach, the selection of this triplet as a function of time is considered as an optimization problem. While many different quantities such as gait speed or the levelness of the back (for transportation of payloads) could be considered as optimization metrics, in this thesis the energy usage was minimized via joint torque measurements. The aim was to increase the amount of time the crawling gait could be performed and reduce the stress on the robot's joints.

4.3.1 Pseudo-static Model

In order to optimize the gait with respect to the joint torques of the robot, a dynamic model of the robot is required. As the Projected Profile crawl gait is conceived as a statically stable gait and performed at slow speeds, the dynamics due to the movement of the robot are not considered to be forces that the motor controllers must counteract. During any part of the gait the robot will not slide if the gaiting direction is orthogonal to gravity and if the robot were to relax its joints it would collapse. Considering this, the resultant joint torques can be seen to be a function of gravity. This pseudo-static model of the robot can then be used to produce cost metric for the optimization procedure. While conceptually simple, analyzing the projected profile closed chain manipulator to

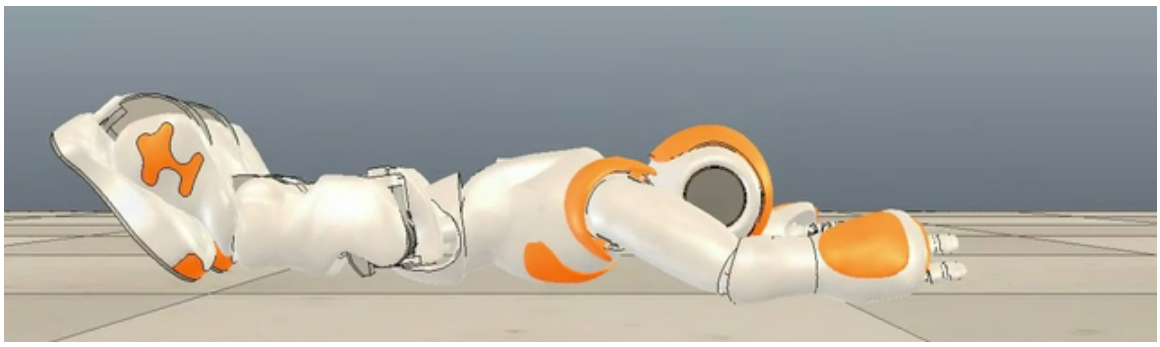

Figure 18: A sample screen capture of the V-REP simulation of the Nao robot in the closed chain configuration. The robot is set to different poses and then the joint torques are read after a short settle time.

produce the system dynamics is challenging. In lieu of this, the robot was simulated for different values of the angle triplet in the closed chain configuration to generate a table of torques. This table of torques is then interpolated to produce a model of joint torques as a function of the parameterized angles:

$$[\tau_2, \tau_2, \tau_4, \tau_5] = jointTorques(\theta_3, \theta_4, \alpha) \tag{4.21}$$

where $\tau_2, \tau_3, \tau_4, \tau_5$ are the torques of one of the ankles, knees, hips, and shoulders, respectively. In this case, only one of each of the joints is considered because the gait is laterally symmetric. This means the torque of the left joint should be the same of the right joint. τ_1 is not a product of the function as θ_1 is not an actuated joint.

The V-REP simulator by Coppelia Robotics was used to gather joint torque data. It uses the Open Dynamics Engine (ODE) as its dynamics solver and is distributed with a model of the Nao Humanoid Platform. The model of the Nao kinematically corresponds with the Nao V4 and has the same mass values for the links. The Nao V4 is the robot used in this thesis. It has an easy to use API that can interface with C++, Python, or MATLAB.

Figure 18 shows a screen capture of the Nao model in the V-REP simulator. The Nao was configured to be in the closed chain and set to different joints angles. The initial and final values of α were constrained due to the gaiting requirement of the elbow to start at extension and end at flexion. The ranges of θ_3 and θ_4 were defined according to what seemed like plausible knee and hip angles for the gait. Using these limits a discrete set of triplets were defined at a 2.5° resolution for each triplet parameter. These triplets were sent through the kinematics equations to produce the joint commands for the

Configuration Parameter	**Minimum Angle (degrees)**	**Maximum Angle (degrees)**
θ_3	-5	45
θ_4	15	-30
α	-30	-90

Table 1: Table of initial and final joint angles for each angle in the configuration triplet used to generate the set of angles used to configure the simulated Nao robot. The resultant set had 9,975 configurations with an angular resolution of 2.5°.

simulated Nao robot. Table 1 lists the parameters that describe the triplets tested. In total, 9,975 different joint configurations were simulated. To allow for the effect of any dynamics generated by the change in configuration to settle, the torque values of each of the joints was recorded after a period of one second.

4.3.2 Optimize ¡——— Everything after this still needs to be edited

As a requirement of the gait, the initial and final poses for the closed chain phase needed to be common to the open chain phase of the gait.

Now that we have things set up and we have our model/function/table thing we can try to optimize the gait. The method of optimization we used was a genetic algorithm on cubic splines. We knew where the parameters have to start and end so we set the cubics terminations there. We told it to do it in 1 second because we knew that would work as the crawl with the constants works. Optimization cost was this: ... which we weighted according to which joints were stronger. We threw this into a basic genetic algorithm which tweaked the 12 parameters (4 for each configuration spline) and iterated on this X many times.

REFERENCES

[1]

[2] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*. New York, NY, USA: Cambridge University Press, 2000.

[3] *The Warrior Ethos and Soldier Combat Skills, FM 3-21.75*, Headquarters, Department of the Army, 2008.

[4] V. Zatsiorsky and B. Prilutsky, *Biomechanics of Skeletal Muscles*. Human Kinetics 10%. [Online]. Available: https://books.google.co.in/books?id=THXfHT8L5MEC