Free LittleDog!: Towards Completely Untethered Operation of the LittleDog Quadruped

Michael N. Dille

CMU-CS-07-148
August 2007

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Christopher G. Atkeson, Chair
J. Andrew Bagnell

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

This research was sponsored in part by the DARPA Learning Locomotion Program and a Siebel Scholars Fellowship.

Keywords: Legged autonomy, Quadruped vision, Quadruped perception, Odometry, Learning Locomotion, LittleDog

Abstract

The LittleDog robot is a 12 degree-of-freedom quadruped developed by Boston Dynamics and selected for use in the DARPA Learning Locomotion program, in which machine learning is applied to develop controllers capable of navigating rocky terrain. Presently, it is typically constrained to operate within wireless range of a host desktop computer and within a fixed workspace surrounded by a motion capture system that globally localizes the robot and specially marked terrain boards without the use of onboard sensing. In this thesis, we explore a variety of strategies for expanding the capabilities of this platform in the theme of relaxing these operational constraints and with the goal of allowing operation in arbitrary locations outside of the fixed workspace and without a host computer. Towards this end, we start by addressing the straightforward technical issue of physical independence by demonstrating a viable onboard controller in the form of a compact single-board computer. Next, we attempt to resolve the lack of onboard sensing through computer vision by attaching a camera to the robot and developing the necessary procedures for calibrating it, synchronizing its data stream with existing state data, and compensating for the additional weight of the camera. Using this, we demonstrate mapping and navigation of terrains outside the motion capture system containing both planar and simple structured three-dimensional obstacles. In conjunction with this, we develop and implement several dead reckoning strategies, one including a complete kinodynamic model of ground contact, to compute odometry information enabling reasonably accurate continuous pose estimation. Finally, we complete a brief exploration of alternatives for local sensing and reason about extensions to more unstructured environments.

Acknowledgments

I wish to express my sincere gratitude to the host of individuals who have made the completion of this thesis possible, and just as importantly, fun. First and foremost, to my advisor, Chris Atkeson, for his amazing sense of humor, his constant guidance, and his unfailing patience with my perpetual desire to invent a better wheel; to Drew Bagnell for taking the time from his impossible schedule to serve on my thesis committee; to Sharon Burks, Deb Cavlovich, Catherine Copetas, and Mark Stehlik for guiding me through the process and offering their encouragement along the way; to the other members of the CMU LittleDog lab for always being around to offer suggestions, especially Joel Chestnutt for graciously permitting me to use his footstep planner; to my parents for their endless support; and finally to the fine folks at Boston Dynamics for creating a very pleasant platform to work with and tolerating the abuse our robot endured during its "extracurricular activities" at my hands.

Contents

CHAPTER 1

Introduction

The LittleDog robot, shown in figure 1.1, is a quadrupedal (four-legged) walking robot developed by Boston Dynamics Inc. (BDI). It has three degrees of freedom in each leg – roll and pitch at the hip and a bend at the knee – for a total of 12 overall (shown schematically as coordinate frames in figure 1.2), weighs approximately 3 kilograms, and stands about 30 centimeters tall. Available publicly as a research robot, it has gained some fame for its use in the presently ongoing DARPA Learning Locomotion program. Both in its overall design and especially as used in this program, it possesses little in the way of local onboard sensing and relies extremely heavily on information provided by external sensors if not operating blindly.

In this thesis, we seek to extend the operational environment of the LittleDog beyond the instrumented laboratory environment, ultimately to arbitrary environments, thereby opening great opportunities for autonomy. Towards this goal, we tackle the issues of achieving physical independence from a stationary host computer and the use of only onboard sensors for tracking the position of the robot and objects in its environment.

1.1 Learning Locomotion Program

The Learning Locomotion program is an effort funded by the Defense Advanced Research Projects Agency (DARPA) to accelerate the development of software utilizing machine learning that allows a robot to navigate and traverse difficult terrain. Six

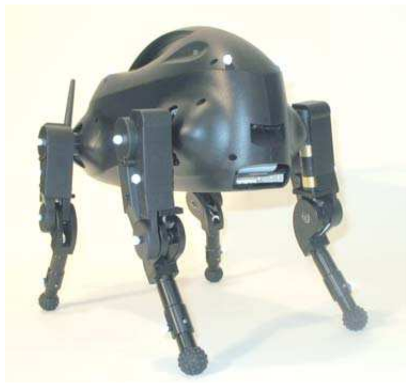

Figure 1.1: Front view of LittleDog robot. Copyright Boston Dynamics.

universities and research laboratories across the United States are participating by simultaneously developing such software, the capabilities of which are then compared against one another and evaluated against baseline metrics.

The LittleDog robot was the platform chosen for this program due, among other factors, to its legged nature lending it the unique ability to step among rocky obstacles as well as the simplicity stemming from its lack of onboard sensing. Indeed, for the purposes of this program, the problem of terrain traversal is simplified by assuming that the robot pose and world map (terrain topography and pose) are globally known. This is achieved by data provided by a commercial motion capture system, which tracks retroreflective markers attached to all objects of interest using a camera array with which it triangulates three-dimensional marker positions. In order for this to function, all objects must remain in the volume visible to at least several cameras. For the Learning Locomotion program, markers were attached to the robot and terrains, and all development and testing is performed within the frame of a truss (shown in figure 1.3) to which the cameras are attached.

1.2 LittleDog Architecture

A schematic diagram of the operational architecture of the LittleDog platform is given in figure 1.4. As this indicates, computation is divided in halves. The first, comprising real-time low-level control tasks such as reading sensor inputs and computing torque outputs, runs on an embedded computer running QNX colocated on the dog. PD control loops run at 1kHz on this computer, which is treated by teams as a closed, opaque module. This communicates via wireless 802.11b ethernet with

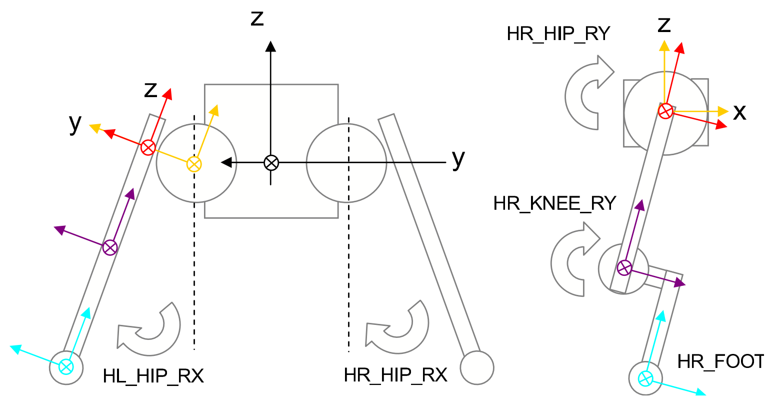

Figure 1.2: Schematic of coordinate frames at each joint degree of freedom. Copyright Boston Dynamics.

a host desktop computer running Linux, on which teams develop and execute their control code, which sends commands to the dog's computer at 100Hz. The motion capture system communicates directly with the host computer, delivering object pose data at approximately 120Hz.

The simplifying assumptions of having a known world state are strongly reflected in this architecture. Other than the optical encoders reporting the angle of each joint and instruments measuring values such as battery voltage, few useful onboard sensors are present. Single-axis force sensors which are located at the base of each foot and measure force up the leg necessarily return values that depend on the angle of contact with the ground. On top of this, they drift and are extremely noisy, making it difficult to implement even a reliable ground contact detector. Likewise, while the commodity infrared proximity sensor installed in the front of the robot works reliably, its very limited range (approximately half a meter) and resolution (just a single scalar value representing distance to the nearest object) allows it to do little more than detect the presence of a large obstacle ahead or the edge of a table on which the dog may be walking. Finally, there is an onboard inertial measurement unit (IMU) built from a set of accelerometers and gyroscopes that together provide estimates of body orientation, angular velocity, and linear acceleration. This sensor is generally fairly accurate, though it too is subject to some amount of drift over time. However, it naturally cannot provide information about the environment, and using it to track the robot's position (e.g., by twice integrating the acceleration it reports) is simply not practical due to rapid error build-up.

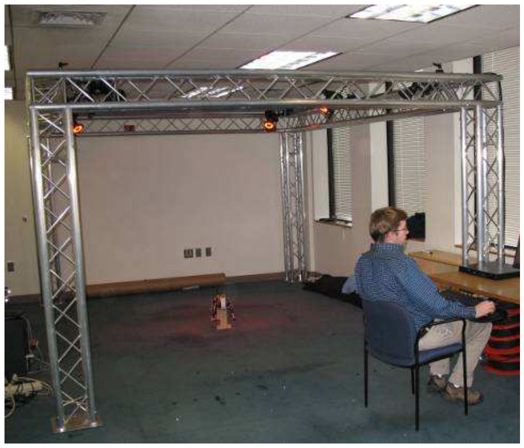

Figure 1.3: The frame of the motion capture system within which the dog must remain for it to operate. Copyright Boston Dynamics.

1.3 Thesis Goals and Contribution

The grand goal of this thesis is to extend the capabilities of the LittleDog robot to environments outside the laboratory, ultimately to arbitrary environments. This can be broken down into roughly two subgoals:

1. Completely untethered operation.

   This is the task of making the robot physically independent, without reliance upon a fixed host computer. In practical terms, this is the engineering challenge of developing and attaching an onboard high level controller in its place. We will describe how we accomplished exactly this in a succeeding chapter.

2. Completely local estimation of robot pose and environment state.

   This is the task of attaching appropriate local sensors to the robot as well as leveraging any existing or intrinsic sensing to acquire as much state information as possible without the motion capture system. Towards this goal, we will describe my efforts at computing odometry during walking and taking a computer vision approach to the environment-sensing problem, each of which we will also describe in a respective succeeding chapter.

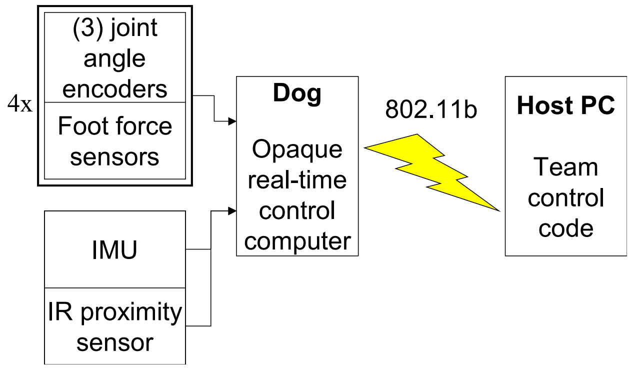

Figure 1.4: Schematic of the LittleDog operational architecture.

CHAPTER 2

Related Work

The LittleDog robot is one of a number of legged robots that have been developed. As can be seen from momentarily forthcoming descriptions of a number of these, achieving a significant level of autonomy – requiring both physical independence and local sensing – is a common goal.

As the design of each of these robots is typically rather complex, often proprietary, and different from the others, a discussion of the details of how each achieves physical independence or might compute odometry is neither very practical nor enlightening. Instead, one may note that in most of these cases, vision-based sensing was chosen to enhance autonomy as it was for this thesis. This is no doubt influenced by the biomimetic desire to duplicate the primary sensing mechanism of the most familiar and developed legged organisms such as humans and dogs. It is also surely a pragmatic result of the fact that cameras are inexpensive, lightweight, and well studied as a sensor for mobile robots.

2.1 Humanoids

2.1.1 H6 and H7

These two robots, of which H7 is shown navigating an obstacle field in figure 2.1, are humanoid robots developed at the JSK Laboratory at the University of Tokyo, Japan. It was designed to be self-contained with batteries and control computers onboard, though it is often run tethered with a harness to protect against falls.

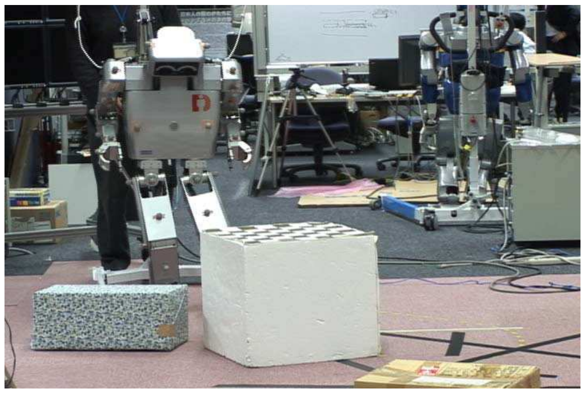

Figure 2.1: H7 navigating an obstacle field. Reprinted from [12].

This robot's head is equipped with a stereo camera, which has been used for a variety of applied vision research efforts. Among these are work on 3-D map building using visual odometry [12] [15] and building height maps from stereo images for navigation [5].

2.1.2 HRP-2

This robot, developed in Japan by the National Advanced Industrial Science and Technology (AIST) and Kawada Industries, is also designed to be self-contained, though it too is usually run tethered with a harness.

Its head contains a 3-camera stereo system that has been put to uses similar to those performed on H7, and this has often been augmented by operating with a motion capture system not unlike the one employed for the LittleDog robot. An example (depicted in figure 2.2) of this includes [9] in which the robot's internal cameras as well as external cameras and rangefinders were tracked by the motion capture system and used to build global planar maps and height maps (respectively) of the terrain, which in turn were used by a footstep planner for navigation. This is indeed roughly the very procedure we will later implement on the LittleDog robot, though with an attempt at using locally-derived pose estimates rather than those provided by a motion capture system.

Another vision-based use of HRP-2 [8] that builds on that work involves 3-D tracking of known obstacles (whose geometric models are stored in a library) using

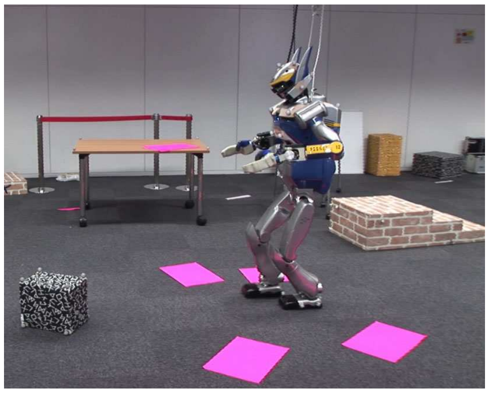

Figure 2.2: HRP-2 navigating a planar colored obstacle field. Reprinted from [9].

edge-fitting and using their tracked positions to build 3-D models of the world that can then be navigated. An example of this includes step-climbing (also depicted in that figure), in which objects are assumed to remain fixed and the body position is tracked relative to them without a need for the motion capture system used in the previous example.

Still other work has been done in an effort to grant HRP-2 autonomy using local sensing, such as [14], which is a novel application of monocular Simultaneous Localization and Mapping (SLAM) to a humanoid in which the vision data was tightly integrated with the robot's odometry and inertial sensing using an Extended Kalman Filter (EKF).

2.1.3 Asimo

Asimo is another humanoid robot, famously developed by Honda which through various arrangements is available for robotics research. Like the previous two humanoids, it too contains a stereo camera in its head.

A variety of work, largely within Honda, has been done to use these cameras for object recognition with the eventual goal of allowing Asimo to autonomously sense and manipulate objects. In the realm of navigation, [10] (by the authors of the similar publication on HRP-2) presents an application in which an overhead camera tracks the pose of the robot in a field of planar obstacles (an example of which is shown in figure 2.3) through which a footstep planner guides the robot. This is again similar to a procedure we will implement on the LittleDog robot, though in our case the camera is colocated on the robot providing more genuinely local sensing.

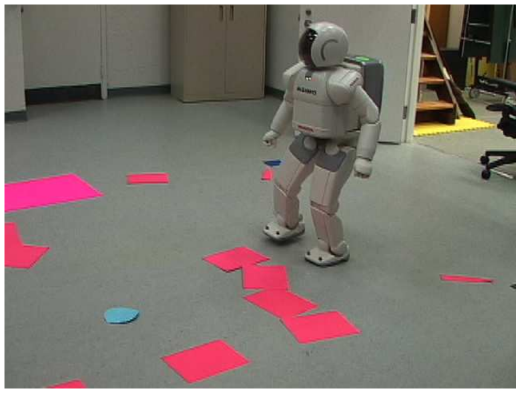

Figure 2.3: Asimo navigating a colored obstacle field. Reprinted from [10].

2.1.4 QRIO

QRIO, pictured in figure 2.4, is yet another humanoid robot, this one developed by Sony. Like the others described so far, it also sports a stereo camera in its head.

Figure 2.4: The Sony QRIO robot. Copyright Sony.

During its development, some highly relevant work [13] [3] was completed to allow it to autonomously navigate unknown environments. Specifically, odometry was used to track the robot's pose in a coarse occupancy grid while moving around in an environment for which its stereo camera was used to continuously construct a map. This is precisely the goal of the vision application described in this thesis, though we use only a single camera.

2.2 Quadrupeds

2.2.1 Aibo

The Sony Aibo, displayed in figure 2.5 is a quadruped originally marketed as a toy that has acquired much publicity for its use in the domain of robot soccer, in which it uses a monocular camera built into its nose to track objects on the field. Through this domain, it has spawned a vast body of work on multi-robot coordination, various aspects of machine learning, and stochastic localization. Because of the limited image resolution and computational power available, objects on the soccer field are colored to make tracking them easier, and many additional advancements in robust color segmentation (such as [17] on which an algorithm described later in this thesis is roughly based) have been made as a result.

Figure 2.5: The Sony Aibo robot. Copyright Sony.

2.2.2 BigDog

The BigDog robot, shown in figure 2.6, is another quadruped developed by BDI, and as its name suggests, it may be thought of as the LittleDog's bigger sibling. It is currently under development partially funded by DARPA with the goal of making an effective pack mule for soldiers, which it could accomplish using person tracking with its onboard stereo cameras. This is an active and somewhat secretive project, and we are not aware of any publications that have yet stemmed from this work.

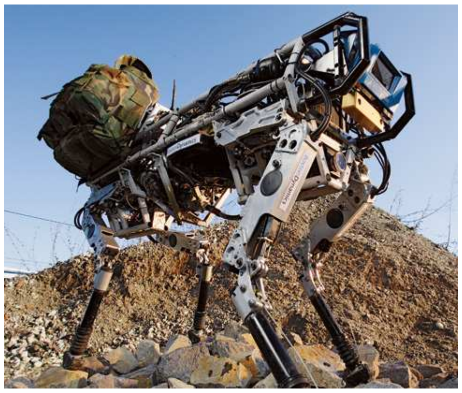

Figure 2.6: The BigDog robot. Copyright John B. Carnett, Popular Science.

2.2.3 LittleDog

The work in this thesis of course builds quite heavily on the development of the LittleDog robot because it extends the platform.

For their part, DARPA and BDI have expressed essentially no interest in expanding the autonomy of the robot by integrating additional local sensing given the irrelevance of such an effort to the Learning Locomotion program. A number of other teams, including Stanford University and the Massachusetts Institute of Technology (MIT), have expressed interest in such ideas. To our knowledge, MIT has experimented with attaching a 1-D laser rangefinder to the robot, and Stanford has worked with a stereo camera. However, we are unaware of a dedicated effort similar to this one in any other team or any publications therefrom.

CHAPTER 3

Achieving Physical Independence

A reasonable definition for physically independent operation of the LittleDog might be that it does so with all computers controlling it and all sensors upon which it is relying being colocated exclusively on the robot. Naturally this implies a significant level of autonomy, but this thesis does not seek to achieve autonomous operation per se since it is convenient to issue high level commands from a remote computer to start, stop, or switch between behaviors. Essentially complete autonomy is then only a very small step away.

3.1 Existing Lack of Physical Independence

As described in the preceding introduction to the LittleDog architecture, the robot uses an internal onboard control computer for real-time low-level tasks such as state data collection and joint-level control. This computer is inadequate for independent operation for several reasons. First, it is designed to be closed to teams, who are to treat it as an opaque module, and does not accept user-provided programs. Furthermore, it has no exposed ports for additional sensors, so even if one could run programs on it, these could make use of only the extremely limited existing onboard sensing. Lastly, it is rather underpowered (roughly comparable to a higher-end Intel Pentium 1), rendering it unsuitable for any significant additional sensor processing such as manipulating images from an onboard camera. Thus, we seek to develop and attach a second higher-level control computer that is able to communicate with the internal one.

Admittedly, this is an engineering problem that is somewhat artificial in that it seeks to overcome a constraint arbitrarily imposed by a third party (namely, the closed nature of the internal computer). However, it is not a completely meaningless one because the technical inadequacies rendering the internal computer inappropriate for heavy additional sensor processing necessitate solving it even if the internal computer were accessible. As such, it represents the development of something new that augments the existing system.

3.2 Development of Onboard Control Computer

For the heart of the high-level control computer, it was quickly decided that it was best to use one of the many existing embedded single-board computers on the commercial market due, among other factors, to time, cost, and skill constraints. While searching for one, the following criteria were developed:

x86 architecture The LittleDog control libraries are provided to each team precompiled for this architecture.

Reasonably fast The computer must be fast enough to run the host control software at 100Hz while processing any new sensor data.

Sensor inputs The computer must have ports to which a camera and possibly other inertial, proximity, or force sensors can be attached.

Compact The computer, batteries, and any additional sensors must fit on the dog.

Lightweight All of these items must not weigh more than the robot can safely carry, for which we established 1kg as a conservative limit.

Low cost The robot would inevitable fall during testing, potentially damaging the computer, and replacing a highly expensive one is undesirable.

Given these constraints, the single-board computer chosen was the PCM-3350 manufactured by Advantech, shown in figure 3.1. This computer adheres to the PC/104 form-factor (less than 10cm square) and boasts a 300MHz Cyrix GX1 processor and 256MB of RAM. A CompactFlash port permits multiple gigabytes of disk storage, enabling the use of a full-blown Linux operating system. Connectivity

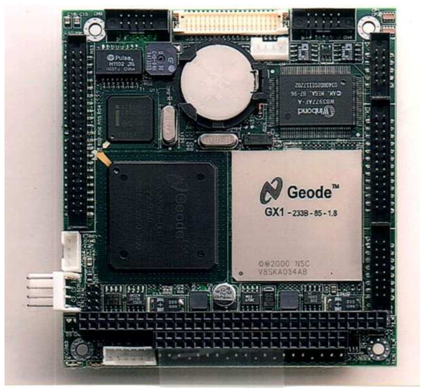

Figure 3.1: PCM-3350 single-board computer chosen as second high-level controller. Copyright Advantech.

proved quite adequate for attaching additional sensors, given two USB port and two serial ports among others.

To power this computer, which requires 5 volts at up to 3 amps, we designed and fabricated a custom power supply circuit board. It was our goal to be able to power the computer using a commodity Nickel Metal Hydride or Lithium Polymer battery pack providing as little as 7.2 volts, and no voltage regulator commonly available at the time possessed these specifications.

In order for this computer to communicate with the internal LittleDog control computer, we purchased a PCMCIA to PC/104 adapter into which we inserted a Lucent Technologies 802.11b wireless ethernet card. By instructing the driver to act in "Master" mode, the robot would see the second computer as a wireless access point and associate with it without the need for an additional external access point. Though this worked well, it later became unnecessary when BDI released a wired ethernet cable that could directly connect the two computers.

3.3 Integration and Testing

To physically adhere the computer to the LittleDog, we built a small wooden saddle that interfaced the curved top of the robot with the flat bottom of the plastic enclosure used for the computer. This proved quite effective, and later efforts reused this saddle for camera attachment.

As a thorough test and demonstration of the capabilities of this platform, we entered the robot in the 2006 Carnegie Mellon Mobot Race [11]. This race is an outdoor sidewalk navigation competition in which a robot must pass through a series of waypoints, most of which are connected by a white line painted on the sidewalk that grants a free (albeit deliberately suboptimal, so as to encourage alternative strategies) path if followed through the waypoints.

In preparation for this application, we attached a commodity Logitech USB camera to the dog and connected it to the high-level control computer. A picture of the entire system is shown in figure 3.2. We also modified an existing trot gait written for the Learning Locomotion program to accept a scalar “steering value,” the sign and magnitude of which would cause the robot to turn left or right to varying degrees during its walk. We then developed a simple image processing algorithm that detected the line using adaptive color thresholding that sought pixels with an intensity at least a fixed number of standard deviations above the average intensity of the several preceding frames. This then computed an offset representing the distance of the line from the vertical center of the image in a small band across the center of the image and fed this as a steering value to the walking controller. Though simple, this algorithm could successfully steer the robot around extremely sharp turns.

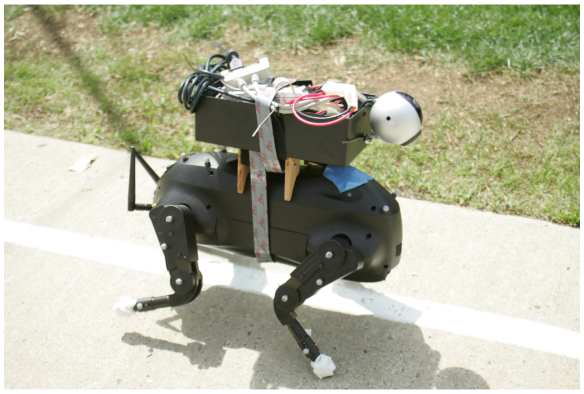

Figure 3.2: Self-contained system with integrated high-level control computer and camera preparing to compete in Mobot race. Photograph by Debra Tobin, CMU.

On race-day, this system followed the line without any problems and passed through course waypoints successfully, as depicted in figure 3.3. It was only tripped up when, on a steep hill present in the course, the robot fell forward because no balance controller was active at the time.

3.4 Conclusions

As described in this chapter, we have successfully achieved physical independence by augmenting the existing robot with a second high-level control computer, and we have also demonstrated a viable outdoor application. This suggests that the LittleDog can indeed be run autonomously, and the rest of this thesis describes how it might do so more effectively.

Unfortunately, even the additional computer we selected suffers from the significant limitation that it is also rather slow. To better quantify this, we ran a simple experiment in which we tracked 10 points through a series of images using the well-known Lucas-Kanade point tracker [16]. This is considered a primitive operation in computer vision insofar as sets of tracked points are then used to determine information about three-dimensional geometry. Yet, on this computer, this operation alone ran at only approximately 1Hz, which is vastly below the 20-30Hz frame rates considered real-time.

The natural and obvious solution to this problem is to simply find a faster computer, and to some extent this can be done given the rapid rate at which new ones are developed. However, faster ones tend to be targeted at less mobile applications and are thus bigger, heavier, and have still greater power requirements. This raises a significant challenge given that the size and mass of the current design can be increased little before it becomes highly unwieldy.

Thus, while we are confident we have demonstrated that physical independence is quite possible, we openly admit that a fast (desktop-sized) host computer is very convenient in practice and that nearly all succeeding work described in this thesis was performed using a desktop host so as to reduce emphasis on optimization during the development process.

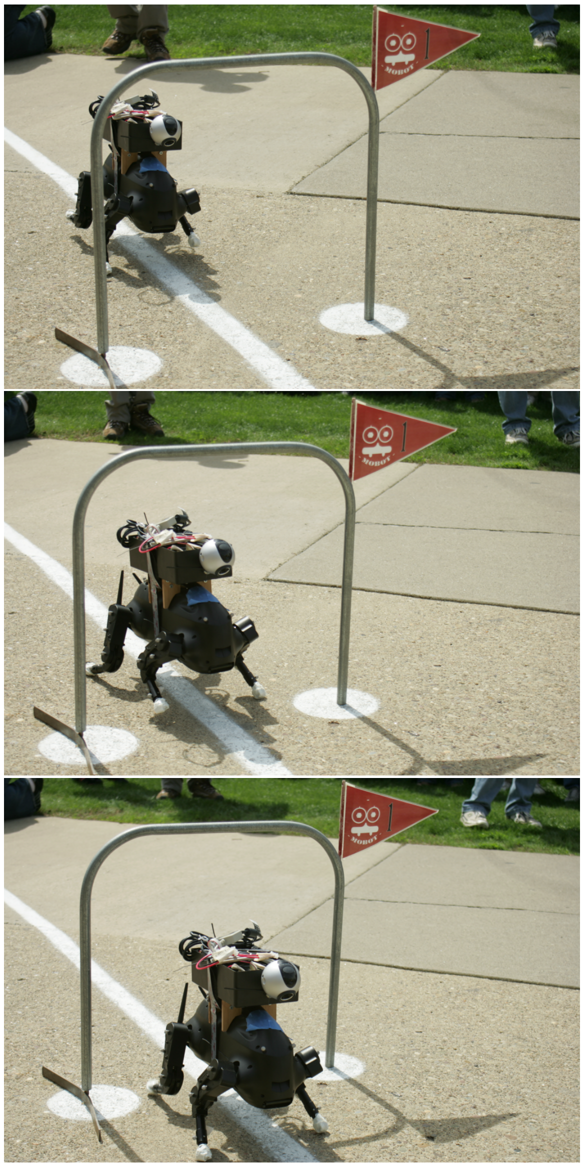

Figure 3.3: Robot passing through a waypoint during Mobot race. Photographs by Debra Tobin, CMU.

CHAPTER 4

Computing Odometry

One of the most fundamental strategies for determining one's position, or localizing oneself, at a given point in time is to track how one has moved from some initial starting point. This process of performing such tracking is called odometry, or more colloquially, dead reckoning. If the world of interest can be described relative to this initial starting point without specific regard for locations in some global coordinate system — as it can be in the case of mobile robots such as this one — then such information can be vitally useful since it represents the complete robot pose state.

In this chapter, we introduce the most intuitive algorithm for computing robot odometry while walking and then present two refinements that better model foot contact with the ground in an effort to improve the accuracy of the tracking. We then present some basic comparisons and conclude by reasoning about fundamental limitations of such tracking and how these might be addressed.

4.1 Odometry I: "Naive" Model

4.1.1 Description

The simplest algorithm for computing odometry follows easily from some intuitive reasoning about the robot's motion. Suppose that at some point, the complete robot pose $T_{body} = \{R_{body}, p_{body}\}$ is known and that it is standing with at least three feet in contact with the ground[1]. Using joint angles provided by each joint's encoder and the known kinematic model of the dog (link lengths and their spatial relationships at each joint), forward kinematics can be applied to determine the relative pose $T_{body \to foot_i} = \{R_{body \to foot_i}, p_{body \to foot_i}\}$ of each foot. The global position T_{foot_i} is then simply $T_{body} \cdot T_{foot_i}$.

Next, we apply the critical assumption that the position p_{foot_i} of each foot on the ground does not change as the dog moves until that foot is lifted (i.e., we assume that stance feet are firmly planted on the ground and act as ball joints whose center is the foot position). With at least three feet on the ground, we can always solve the system $R_{body} p_{body \to foot_i} + p_{body} = p_{foot_i}$ (which is uniquely determined for exactly three feet on the ground and overconstrained for four) for $T_{body} = \{R_{body}, p_{body}\}$. Indeed, this is an instance of seeking to solve for the rigid transform (here, the body pose) between two coordinate systems given pairs of correspondences (here, foot positions). In the case of three correspondences, straightforward vector geometry will grant an explicit analytic solution. In the case of four (or more, in more general scenarios) the problem is substantially harder, though again there exist well-understood algorithms, an excellent survey of which may be found in [7].

Finally, given the body pose, the pose of the flight foot (if one is off the ground) can again be determined by forward kinematics. Once this foot returns to the ground, its position may be recorded and, now that it has become a stance foot, treated as fixed until it again leaves the ground. This process may be iterated *ad infinitum* to perpetually track the robot's position.

4.1.2 Problems

The naive model, as its name suggests, suffers from several problems that reduce its accuracy or otherwise limit its performance. These include:

1. Stance feet do not necessarily stay still, violating the critical assumption that p_{foot_i} is fixed for each foot. These violations may stem from:

   (a) Foot shape.
   The robot's feet are in fact spheres rather than infinitesimal points as the assumption implicitly requires. As the radius ($r \approx 1$cm) is nontrivial, roll about the foot ($r\theta$, which is a very non-negligible 1.6cm for a rotation of

[1]With less than three feet on the ground, the robot is generally not statically stable, and its motion is then governed by additional dynamics that we do not seek to model. We believe that most typical situations will involve at least three stationary feet, trot gaits excluded.

$\frac{\pi}{2}$) is not accounted for and can lead to significant error-buildup across multiple steps.

   (b) Foot slip.
   If friction between a foot and the ground is insufficient and any lateral forces exist, the foot may slip along the ground. Such motion necessarily moves the foot, again violating the critical assumption, and is not accounted for by rolling around the surface of the spherical foot. Further, since the frequency and magnitude of slip depends upon the thrusting force applied by the rest of the leg, slip varies within and between gaits.

2. The estimate of $T_{body \rightarrow foot_i}$ may be erroneous. Since the kinematic model of the dog is very accurate as it is derived from the CAD model from which it was manufactured, such errors are most likely from incorrect estimates of joint angles. These errors in turn may be caused by:

   (a) Encoder error and slip.
   Joint angles are measured using optical encoders that output a pulse for each small rotation increment, which are then counted to track aggregate rotation. If any pulses are missed by the receiving electronics or if the rotational axis mechanically slips without triggering pulses, untracked rotation will occur.

   (b) Gear train backlash.
   The design of the LittleDog includes fairly significant gearing to allow relatively small motors to exert comparatively high torques. Due to various design considerations, joint encoders are located on the motor-side of the gearing. This means that any play between gear teeth, which may be up to several degrees, is not reported in the encoder readings.

3. As described, the body orientation is computed without any regard for data provided by the IMU. Even if such data is of relatively low quality, this still discards information. This may, however, be resolved somewhat readily by taking a sensor fusion approach by combining independent estimates using, for instance, a Kalman filter.

4.2 Odometry II: First Approximation of Rolling Contact

4.2.1 Description

A logical extension to the odometry computation algorithm just presented is to more accurately model foot contact as rolling around a sphere rather than as a ball joint.

As a first step, we must determine some information about the ground plane, specifically its unit normal vector $\hat{n}_{ground}$. This can be done fairly readily any number of ways, for instance by fitting a plane to the foot-center positions at some initial point at which the body orientation is known and it is assumed that a given set of feet (of which there must be at least three to uniquely determine the plane) are in contact with the ground.

Given this, we may again start out by assuming that at some initial point, the complete robot pose $T_{body} = \{R_{body}, p_{body}\}$ (and thereby $T_{footcenter_i}$ for each foot via forward kinematics) is known. Each spherical foot may be treated as an oriented sphere whose axes are aligned with the local coordinate system at the center of the foot, and the point on the foot in contact with the ground may be estimated to be the point on the sphere having unit normal vector $-\hat{n}_{ground}$ (a property of spheres is that given any unit vector, there exists exactly one point on it whose unit normal is that vector).

At a succeeding timestep after some slight motion, the orientation provided by the IMU may be used to estimate the body orientation R_{body}. Through the forward kinematics, we may estimate each foot's orientation $R_{foot_i} = R_{body} \cdot R_{body \to foot_i}$. Each foot may again be treated as an oriented sphere (with this new orientation), and the point in contact with the ground also may again be estimated as previously described. As depicted in figure 4.1, this gives us two points on the sphere separated by some angle θ_{roll}, the great circle arc through which is approximately the path along which the foot rolled on the ground. The distanced rolled along the ground is then simply the arc length $d_{roll_i} = r_{foot}\theta_{roll_i}$ along this circle, and the line on the ground along which this roll took place (with unit vector v_{roll_i}) is the intersection of the plane of the great circle and the ground plane. This may be used to update the estimate of the foot center position by computing $p_{footcenter_{i_1}} = p_{footcenter_{i_0}} + d_{roll_i} v_{roll_i}$.

At this point, we need only solve the system $R_{body} p_{body \to footcenter_i} + p_{body} = p_{footcenter_i}$ for the body position p_{body}. Methods such as those described in section 4.1 may be used, or it may be solved directly as a linear system given that only the body's position (rather than also its orientation) is unknown. Indeed, the updated position estimate of only one foot center is required for this computation, but using

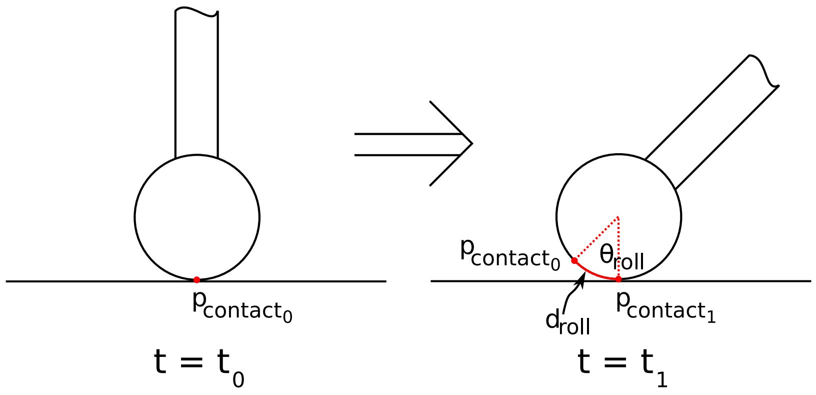

Figure 4.1: Diagram demonstrating how ground contact moves around a spherical foot as it rolls.

the data provided by all stance feet and applying least-squares to the overconstrained system will naturally increase robustness. Lastly, again as with the previous method, the pose of any flight feet can be determined by forward kinematics, and once these return to the ground, they may be treated as stance feet and the process iterated.

4.2.2 Analysis

This algorithm for computing odometry improves upon the previous one in several important ways. First, it much more accurately models ground contact with the robot's feet because it correctly treats them as spheres rather than idealized infinitesimal points. Additionally, it directly incorporates existing body orientation estimates provided by the IMU (or any filtered or otherwise processed version thereof). Furthermore, this version possesses the interesting property that it requires only one stance foot (rather than three) to function, potentially allowing odometry computation during the execution of trot gaits or other fairly dynamic behaviors.

At the same time, however, it does make the assumption that foot contact with each foot moved in a straight line along the great circle arc between the two observed ground contact points on the spherical foot (rather than along some curve that just contained those two points). The effects of this are lessened, though, with high frequency updates (such as at every 100Hz control tick) that might allow for a high-accuracy piecewise-linear approximation of arbitrary curves along the ground. Also, this method relies quite heavily on the IMU (or some combination of other processes to provide a body orientation estimate), without which it simply cannot operate.

Finally, it does not address the previously enumerated problems of foot slip or joint angle errors.

4.3 Odometry III: Kinodynamic Rolling Contact Model

4.3.1 Description

The preceding algorithm is somewhat weak in that it uses state measurements (namely, R_{body} possibly provided by the IMU) to track the execution of a physical process that can be modeled and about which state predictions can be made given known inputs (here, joint motion). Specifically, when at least three feet are on the ground, the robot with its ground contacts can be modeled as a closed kinematic chain in which changes in positions are directly related to changes in joint angles, and thus knowing changes in joint angles allows one to determine these changes in positions. Tracking such position changes then allows one to track the robot's pose.

4.3.2 The Model

We start by assuming that at least three feet are on the ground. We then instantaneously treat each foot as a ball joint centered at the foot center and whose angular velocities correspond to those of the spherical foot. That is, each foot has associated with it three angular velocities ω_{fx_i}, ω_{fy_i}, and ω_{fz_i}. Then select any three non-collinear points on the body (for simplicity, we selected three points in the plane of the hips within the convex hull of the hips) with respective displacements r_j from the body center. For each pair comprised of a stance foot and one of these body points, we treat the segment in between as an independent kinematic chain with 6 degrees of freedom: the three foot angular velocities, the knee angular velocity (ω_{k_i}), and the two hip velocities (ω_{hx_i} and ω_{hy_i}). Schematic side- and top-views of this are given in figures 4.2 and 4.3 respectively.

Taking inspiration from an example of decomposing a closed-loop chain in a similar way as described in [20] (in which two cooperating arms were used for a sawing task), we can compute a 3x6 Jacobian matrix for each independent kinematic chain between foot i and body point j:

$$J_{i,j} = J_i(\theta_{hx_i}, \theta_{hy_i}, \theta_{k_i}, \theta_{fx_i}, \theta_{fy_i}, \theta_{fz_i}).$$

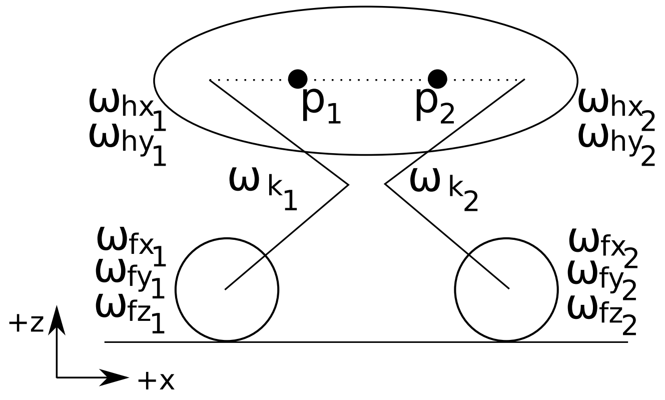

Figure 4.2: Side-view of robot annotated with parameters of rolling contact model.

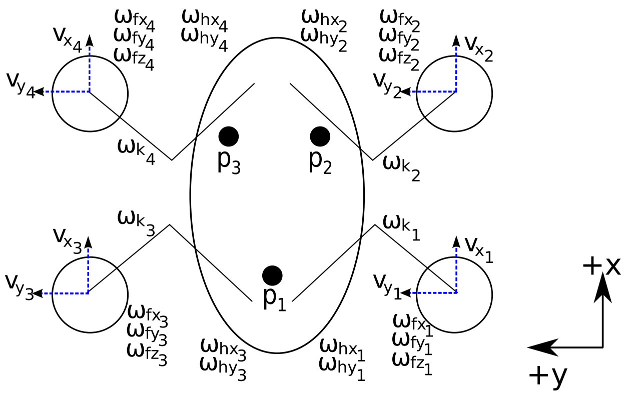

Figure 4.3: Top-view of robot annotated with parameters of rolling contact model.

Then, the body point velocities are given by

$$v_j = J_{i,j} \cdot [\omega_{hx_i}, \omega_{hy_i}, \omega_{k_i}, \omega_{fx_i}, \omega_{fy_i}, \omega_{fz_i}]^T = J_{i,j} \cdot \Omega_i.$$

It is also convenient to define the linear velocity of the body center p_c as $v_c = [v_{c_x}, v_{c_y}, v_{c_z}]^T$ and the angular velocity vector of the body as $\omega_c = [\omega_{c_x}, \omega_{c_y}, \omega_{c_z}]^T$.

Next, we may note that ω_{hx_i}, ω_{hy_i}, and ω_{k_i} are known quantities or can be estimated online (e.g., via an observer that acts as a velocity filter or possibly as simply as computing $\frac{\Delta\theta}{\Delta T}$ at each timestep). We then seek to solve for the unknown foot angular velocities. We can do this by constructing a system of linear equations involving all these quantities.

Consider for now just the case of three feet on the ground. In this case, the unknowns are three triplets of $\omega_{foot_i} = [\omega_{fx_i}, \omega_{fy_i}, \omega_{fz_i}]^T$ for the feet on the ground and three triplets of $v_j = [v_{x_j}, v_{y_j}, v_{z_j}]^T$ for the three body points, giving 18 unknowns in total. For constraints, we can use three triplets of $v_1 = J_{i,1} \cdot \Omega_i$ (separated so that ω_{foot_i} and v_1 are the unknown parameters) to ensure that the velocity of the first body point is consistently predicted, two triplets $v_2 = J_{h,2} \cdot \Omega_h$ and $v_3 = J_{k,3} \cdot \Omega_k$ (where h and k are the two feet respectively nearest to body points 2 and 3) for the velocities of the two remaining body points, and three pairwise equations $v_a \cdot (p_a - p_b) = v_b \cdot (p_a - p_b)$ (for all pairs (a,b) of body points) constraining the body points to move as a rigid body, giving a matching 18 equations in total.

Having solved for the velocity of each body point, the velocity of the body center may be estimated as $v_c = \frac{v_1+v_2+v_3}{3}$ and its angular velocity from the cross products of the body point velocities relative to v_c: for each, $\omega_c \times (p_i - p_c) = v_i - v_c$. However, for the purposes for which we used the model, only the foot angular velocities were of actual interest to us.

If four feet were on the ground, three more unknowns (its foot angular velocities) and three more constraints (relating these via the Jacobian to the linear velocity of p_1) would be added. Intuitively, it might then be desirable to add a fourth body point (resolving excess constraints via least-squares as necessary) to smooth out asymmetries, but it is unclear how to precisely quantify the effects of this.

This model thus provides a rather powerful ability: given joint velocities and knowledge of which feet are on the ground, the body velocity and angular velocities of the body and all stance feet are derived. Integrating these velocities forward allows for the prediction of future foot and body positions, and precisely this may be done to use this model for odometry purposes. Several frames from a simulation in which this model is used to iteratively compute body and foot positions given only a series of joint angles and the knowledge that all four feet were on the ground is shown in figure 4.4.

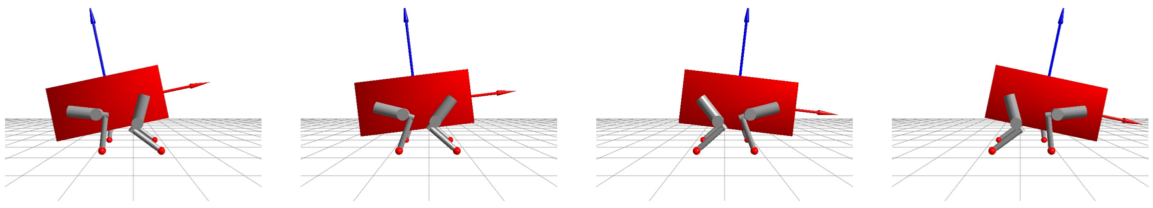

Figure 4.4: Visualization showing model computing body and foot poses from joint angle trajectory.

4.3.3 Applying the Model

To use this model for odometry purposes, we again assume that at least three feet are on the ground and that at some initial point, the body pose T_{body} (and thus each foot pose T_{foot_i}) is known. Further, we now assume that some estimate of the joint velocities is available. Feeding all of this information to the model computes, among other values, the angular velocities ω_{foot_i} of each stance foot. Under the coordinate system shown in figures 4.2 and 4.3 and constraining stance feet to only roll along the ground, we may then derive each stance foot's velocity $v_{foot_i} = [v_{foot_{i_x}}, v_{foot_{i_y}}, v_{foot_{i_z}}]^T = [r_{foot}\omega_y, -r_{foot}\omega_x, 0]^T$. Integrating the foot position using this velocity gives a new estimate of the foot position. Finally, as done in the previous two methods, the body pose and the position of any flight leg may be derived.

4.3.4 Analysis

Computing odometry using this model is technically superior to the previous methods because it integrates an understanding of the actual physical process and functions in a predictive rather than reactive way. Further, unlike the immediately preceding algorithm, it does not rely on already having an accurate estimate of the body orientation.

It remains imperfect, however, since it is still an approximation by virtue of integrating potentially noisy joint velocities to derive succeeding estimates of foot position. Also, it disregards any information provided by the IMU, though as previously noted, the two estimates of body orientation may be combined via optimal estimation techniques. Lastly, it continues to fail to address errors introduced by foot slip or joint angle errors.

4.4 Implementation and Comparison

All three strategies for odometry presented in this chapter were implemented on the actual LittleDog robot and tested during a series of trials in which the dog was instructed to walk forward in roughly a straight line using walking controllers with slightly varying parameters.

Naturally, during the course of such trials, tracking error would accumulate, and the pose estimate provided by the odometry computation would diverge somewhat from ground-truth, which the data from the motion capture system was treated as providing given its historic accuracy to within very sub-centimeter levels. During our testing, we observed that the growth rate of such errors varied rather greatly depending upon the terrain material and the many parameters of the walking gait in use. We did not attempt a formal study of the relative effects of each of these and their many subfactors, but the results displayed in figure 4.5 may be considered quite typical.

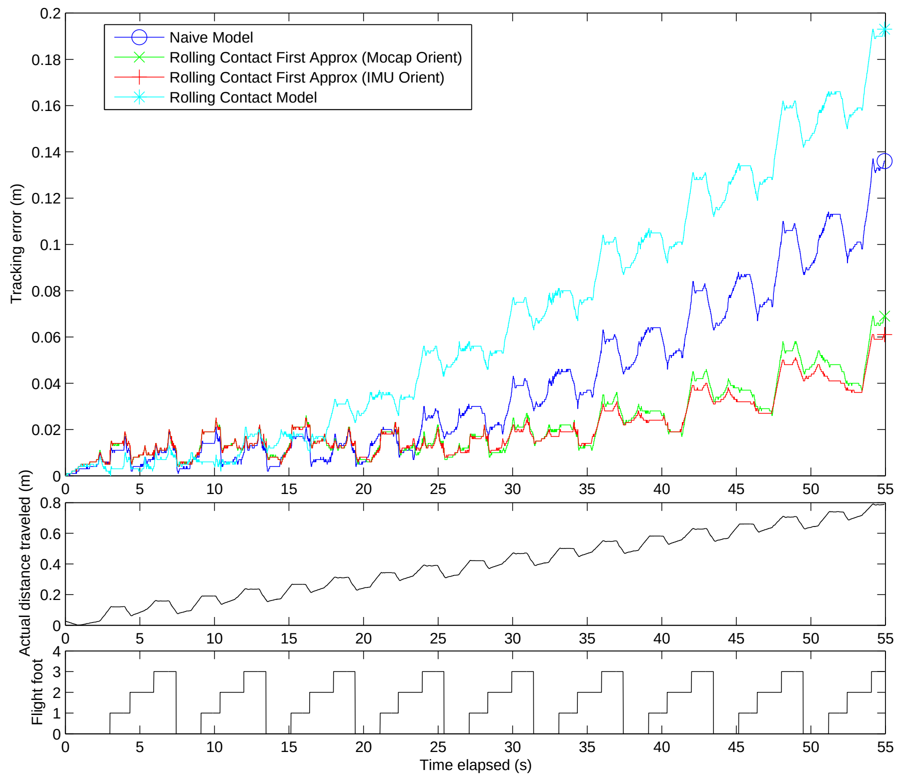

Figure 4.5: Comparison of position tracking error of odometry algorithms during 40-step trial.

This plot depicts the position error (Euclidean distance between ground-truth and the tracked estimate) of each of the three algorithms described in this chapter as they executed during a 40-step sequence with an individual step length of 8cm. Two variations of the second algorithm are presented: one which uses ground-truth motion capture data to derive the body orientation used at each step, and another which relied exclusively on the onboard IMU to provide this.

Our experiments provided a number of interesting results, most of which are summarized in this plot. First, as we expected given its more realistic assumptions, the second odometry model proved to be significantly more accurate than the first model, typically exhibiting half the position error. In this given trial, the prior had accumulated just over 6cm of position error after 40 steps, while the latter had reached nearly 14cm.

As this plot also shows, dependence on the IMU did not represent a significant source of error, since implementations respectively utilizing the IMU and motion-capture-provided orientations had roughly comparable error magnitudes. Occasionally, as is also the case in this plot, the version using the IMU accumulated error *less* rapidly than the version using the motion capture data, suggesting (under the assumption that the motion capture data is indeed strictly better) that other error sources of a significantly greater magnitude are at work.

A somewhat unexpected result was that the third model proved even less accurate than the first, decidedly naive, model, reaching over 19cm of position error after 40 steps. The reasons for this may be numerous (including, we concede errors in its rather complex implementation, which we believe unlikely due to successful unit testing of its individual modules and its performance in simulation), but the most likely culprit we perceive is error buildup stemming from the constant integration of fairly noisy velocity estimates — technically, Euler integration of a function with high-magnitude derivatives. Though we believe that it may be possible to reduce the impact of this somewhat by using a less numerically sensitive technique to update the position estimate at each step, we have not explored such avenues and suspect that the velocity-based nature of this model may represent a fundamentally serious weakness. It bears mentioning, however, that for over 10 seconds of this trial (approximately 5 steps), the accumulated error from this model was generally below that of any of the others. This suggests that this model may in fact be very useful for applications requiring only short-term incremental tracking estimates.

4.5 Conclusions

This chapter described algorithms for computing odometry on the LittleDog while walking, which we implemented and provided at least a rudimentary comparison between in the previous section. We now reflect on fundamental issues surrounding such estimation and attempt to reason about its true practicality.

First, tracking the robot's pose through odometry is an inherently iterative process. Thus, any errors that work their way into the estimate will only be compounded with the passage of time and will never be corrected unless negated by other unmodeled effects. Therefore, whenever possible, short-term differential estimates should be used as they will likely be more trustworthy. In order to reset or at least reduce accumulated error, some form of filtering or other sort of optimal estimation should be used to combine tracked posed estimates from odometry with data provided by other sources such as the IMU or visual tracking from a camera. This was not done in the work performed for this thesis, but it almost certainly should be in any serious application.

Several obvious error sources remain untreated. The first of these is slippage between an otherwise-stationary stance foot and the ground. As discussed in the previous section, this appeared to have a very significant effect on the quality of pose estimates. To make matters worse, it varies greatly depending on the terrain (through its frictional coefficients with the foot material) and the walking gait in use (depending upon its jerkiness and lateral forces applied). These are, however, the two parameters often most easily changed: the material from which the terrain or foot is made can be changed, and the gait can be reprogrammed to behave arbitrarily differently. Ultimately, we would like to sense the occurrence of slip (which would at least allow for a quantifiably reduced confidence in the tracking estimate thereafter) and, if at all possible, to measure it. Sadly, it's not clear that either of these can be done well. Intuitively, simple heuristics can be applied to the problem of detecting slip such as watching for rapid foot accelerations or velocities when none are expected or sudden tilt that might indicate that a foot has given way. With better force sensors at the feet and perhaps a more sensitive IMU, enough data might be available to make progress towards reliably detecting slip and perhaps even modeling its magnitude.

The other primary error source previously mentioned was an erroneous estimate of joint angles. While we have on occasion observed outright encoder error (sometimes possibly due to slip), especially after the robot has suffered a significant fall, this is infrequent and its impact relatively minimal because recalibrating eliminates it. Backlash, however, is a much more severe problem. It varies over time (especially, increasing as the mechanism wears), differs between feet (probably due to slight construction variation and asymmetric wear patterns), and can be completely different across robots (again likely due to construction variation). To some extent, it can be modeled (for instance, by inserting variable angle offsets depending on the estimated direction and magnitude of force on the leg), but doing so accurately is extremely challenging, and we did not attempt it for this thesis.

Overall, we have demonstrated several viable algorithms for at least roughly tracking the robot's pose during motion. As we did not augment this with an additional process that combined this estimate with any other to reduce uncertainty, the quality of estimates provided was generally insufficient for the remainder of the work presented in the following chapter. Thus, ground-truth data from the motion capture system was generally used for later development so as to better isolate sources of error and evaluate the standalone quality of the results stemming therefrom. However, given the fairly high short-term accuracy of this tracking, we are exploring the possibility of building it into the Kalman filter used to estimate the robot's pose during Learning Locomotion program trials.

CHAPTER 5

Mapping and Navigation with Vision

As indicated to earlier, we chose computer vision as our primary strategy for local sensing. This was done for the same reason that many others do so: cameras are compact yet can provide a huge amount of data. In this chapter, we describe the process of integrating a camera as an additional sensor on the robot and our efforts to use it to locally map and navigate the surrounding environment.

5.1 Design Philosophies

A primary understanding with which this problem was approached was that we would not expect either to implement state-of-the-art vision techniques nor to attempt to advance the field of computer vision in any significant way. As the goal of this thesis is to more generally grant the robot greater autonomy, we instead merely sought to apply vision to this platform towards this end.

As such, we made use of the very helpful Intel Open Source Computer Vision (OpenCV) library [4] wherever possible rather than reimplementing many of the standard algorithms, however instructive that might have been. Nevertheless, in quite a few instances, its functionality proved lacking, and much often-laborious reimplementation was required.

It should also be noted that early on in the development process, it was decided that only a single camera (monocular rather than stereo vision) would be used. This route was chosen because two cameras were simply too heavy and bulky to be attached to the dog, and any reasonable combination of size, resolution, quality, and affordability could not be found among commercial stereo cameras available at the time. This somewhat limited the vision strategies available, but naturally much remained possible.

5.2 Camera Integration

Even before anything interesting can be done with camera image data, the problems of selecting an appropriate camera, attaching it, calibrating it (geometrically), and merging the data it provides with the existing state stream must be solved. Though such tasks are often considered uninteresting preparatory work, the significant time invested in their completion and the somewhat unique way in which they apply to this platform merit describing them in at least some detail.

5.2.1 Camera Selection and Attachment

The very first task was the selection of an appropriate camera. Much as with the selection of an appropriate additional onboard control computer, the constraints imposed by our specific scenario engendered several criteria:

Compact The camera and its mount must fit on the dog.

Lightweight The camera and its mount must not weigh more than the dog can carry without significantly losing its agility.

Durable During testing, the dog would inevitably suffer occasional falls, and the camera must neither break nor badly lose its calibration when this happens.

Low cost Good cameras can be arbitrarily expensive, and we chose several hundred dollars as a reasonable limit.

Color We wanted to be able to simplify perception by using color cues in the environment, so a color camera was necessary.

Specifically omitted from this list are requirements for very high framerate or resolution, neither of which were necessary for this application.

Two cameras were selected that roughly fulfilled these criteria:

- Logitech Quickcam Pro 4000
  This USB camera provides 640x480 color images at 15 frames per second (fps). It used an only somewhat refocusable built-in lens.

- Imaging Source DFK21F04
  This IEEE1394 (Firewire) camera provides 640x480 color images at 30 fps and provides a standard CS-type lens mount. We attached a highly adjustable varifocal lens for maximum flexibility.

The saddle constructed to support the additional control computer was reused for attachment of a camera mount on a 14cm boom as shown in figure 5.1.

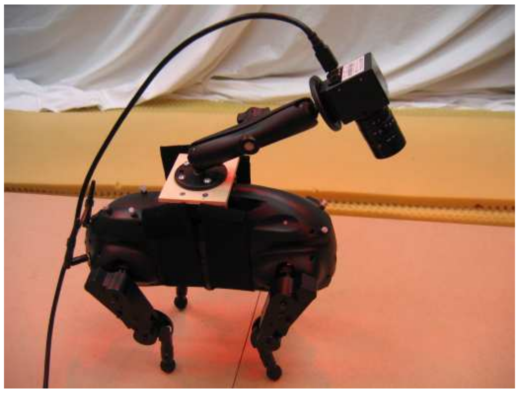

Figure 5.1: Firewire camera attached to LittleDog via boom on saddle.

5.2.2 Camera Calibration

The next step required calibration of the camera by solving for values of parameters in an imaging model that would then allow relating positions of objects in an image to positions in the world. The model used was the very typical pinhole camera model (well described in standard textbooks on the subject such as [2]) with an additional second order radial and tangential distortion model, which was necessary especially when using the varifocal lens because it offered a wide angle at the expense of adding distortions.

This model functions in the following way:

1. Points in the real world are transformed into a local camera coordinate frame:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{camera} = R \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{world} + T$$

The matrix R and vector T are commonly referred to as the extrinsic parameters.

2. Points in the local camera frame are projected onto a virtual image plane unit distance from the origin:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix}$$

3. Projected points on the virtual image plane are warped slightly (modeling lens distortion):

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x'(1 + k_1 r^2 + k_2 r^4) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\ y'(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \end{bmatrix}$$

where $r^2 = x'^2 + y'^2$.

The parameters (k_1, k_2) are the radial distortion coefficients, and (p_1, p_2) are the tangential distortion coefficients. These, with the additional parameters below, are called the intrinsic parameters.

4. Distorted projected points are focused and centered to create the final image:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{image} = \begin{bmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{bmatrix}$$

The parameters (f_x, f_y) are the horizontal and vertical focal lengths (separate to account for non-square pixels on the image sensor), and (c_x, c_y) simply translate the pixels to place the origin at a corner of the image. These four parameters along with the distortion coefficients constitute the intrinsic parameters.

The intrinsic parameters are so named because they result from internal properties of the camera and do not vary based on camera location. The extrinsic parameters, meanwhile, express the pose of the camera in the world and therefore change with it.

To calibrate the intrinsic parameters, the standard technique of capturing frames of a chessboard of known geometry (such as that depicted in figure 5.2) was used. The corners of the chessboard represent a set of correspondences between image points and world points, and feeding these to a camera calibration function, such as that provided by OpenCV, allows one to solve for the complete projection parameters for each frame and optimize for the best overall intrinsic parameters. When the world points corresponding to the chessboard corners were reprojected using the model, sub-pixel error was typical, demonstrating that the model fit well.

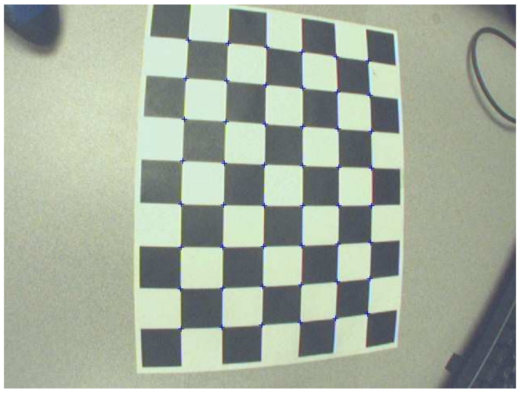

Figure 5.2: Example image captured during intrinsic parameter calibration of the camera, with detected chessboard corners highlighted.

Calibration of extrinsic parameters was then needed to relate information provided by the camera to the coordinate frame local to the robot's body. Specifically, we needed to solve for $T_{body \rightarrow camera}$ representing the pose of the camera relative to the body. While in theory this could be measured physically, this was not deemed practical as it could not be done with any great accuracy and would change slightly each time the camera had to be removed from the robot for Learning Locomotion program development.

Therefore, a somewhat novel technique taking advantage of the motion capture system was developed. First, a custom calibration rig instrumented with motion capture markers was built. This allowed a chessboard to be precisely tracked in the motion capture frame in the same way as the robot. Then, a simple "exercise" trajectory was written to command the robot to wiggle around while capturing a series of typically 30 frames containing the chessboard while the motion capture system tracked both the robot and calibration rig. A snapshot of this process is shown in figure 5.3.

For each frame in the sequence captured, the motion capture system provided (relative to its coordinate frame) a pose T_{body_i} for the body and a (constant, since it remains stationary) pose T_{board} for the chessboard. Given the previously determined intrinsic camera parameters, applying a similar OpenCV calibration function to that used for intrinsic calibration solves for $T_{board \rightarrow camera_i}$. The position of the camera

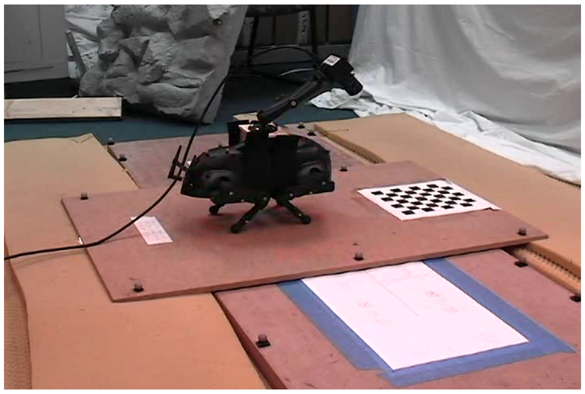

Figure 5.3: Snapshot of extrinsic calibration procedure.

relative to the motion capture system origin is then $T_{camera_i} = T_{board} \cdot T_{board \to camera_i}$. This provides a per-frame estimate of the body-relative camera pose $T_{body \to camera_{(i)}} = T_{body_i}^{-1} \cdot T_{camera_i}$.

To obtain the highest-possible accuracy estimate of $T_{body \to camera}$, each $T_{body \to camera_{(i)}}$ was averaged[1]. Since occasionally not all the chessboard corners in a given frame were correctly detected or the calibration function failed to converge on a solution, causing some values of $T_{body \to camera_{(i)}}$ to be erroneous, RANSAC [18] was applied to provide outlier elimination. Specifically, three randomly chosen estimates were chosen at a time, averaged, and the number of inliers (here, the number of other estimates which, when used to recompute the positions of test set of world points, agreed to within approximately a centimeter) computed. The set of three with the most inliers from a large sampling was found, and the overall estimate computed as the average among all those inliers. In case no set of three found at least 40% of the other estimates to be inliers, failure was returned. Since the quality of the estimate provided by each frame was rather binary (either very good or very poor), this algorithm proved to work very well without any tuning of the RANSAC parameters (such as the size of each sample, the threshold to be considered an inlier, etc.) and allowed for easy calibration without manual intervention.

[1]In our case, this was performed in an ad-hoc way by separately geometrically averaging the translation vectors and using Slerp [19] to average the rotations. This was adequate for us due to the generally small difference in the transforms being averaged, but a better-reasoned method such as the dual quaternion methods described in [6] could as well have been applied.

5.2.3 Latency Calibration and Compensation

The final step in camera preparation required synchronizing the flow of image data with the rest of robot state provided by the internal control computer so that computations involving values from each could be coherently performed. Specifically, this meant solving for the relative latency between the streams so that any necessary compensation could occur.

In the case of robot state data, some latency is introduced during processing on the internal control computer, variable latency is added during wireless transmission (during which packet delays and losses may occur), and then more variable latency injected in the receiving software on the host computer. Fortunately, all state data is marked with the timestamp on the robot at the time of its transmission, offering immunity to this variability. On the host side, image frames must be captured by the camera's electronics, transferred over the USB or Firewire bus, and then pass through several layers of software processing before being received by the control software. Though most of this process is also subject to variability (mostly due to the kernel scheduling the load imposed by the rest of the controller), here too timestamps are available. In the case of USB cameras, the best that can be done is generally a `gettimeofday`() system call in the lowest user-level capture code to acquire a host timestamp. For Firewire cameras, the kernel is able to provide a host timestamp at the time of the isochronous DMA transfer of the image frame, which was found in our case to have jitter in the mere microseconds. Since both streams provide timestamps, they can be synchronized despite variable latency if only the offset between these timestamps can be determined.

The procedure we developed to compute this offset entails standing the robot in front of a large chessboard (to provide a view rich with strong corner features) as depicted in figure 5.4 and tracking a set of corners near the center of the image across frames using the Lucas-Kanade point tracker provided by OpenCV. We then command it to sway from side to side recording the centroid of foot positions relative to the body (from the primary state stream), the centroid of the visually tracked corner positions (from the image stream), and the timestamp from each stream. The normalized square of the distance of both centroids from their respective starting position is simultaneously plotted against the robot state timestamp on the same set of axes, an example of which is shown in figure 5.5. Using an automatic procedure, the phase offset between these two curves is found, giving the the timestamp offset between the streams. For the USB camera we used, this procedure computed a lag of approximately 60ms (or about one frame interval at 15fps) behind the robot state, and a lead of approximately 30ms (or again about one frame interval at 30fps) for the Firewire camera.

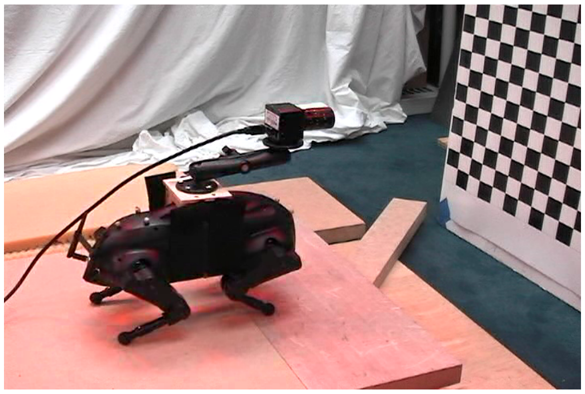

Figure 5.4: Snapshot of latency calibration procedure.

To make use of this offset to compensate for (variable) latency, the robot state at each tick of the controller was saved in a state history array. Then, when a newly processed image frame arrived, its timestamp (adjusted by the calibrated offset) could be looked up in the history array, providing exactly the robot state corresponding to the view presented in that image frame.

Somewhat amusingly, such compensation proved to be vital in our configuration because of a bug in the Firewire driver in the particular kernel used on the host system that caused duplicate frames up to as old as one second (properly tagged with a timestamp as such) to occasionally be returned. Without compensation, this could introduce near-catastrophic consistency errors in, for instance, environment maps generated with these frames. More practically, in a test in which a fixed patch of ground was continuously rectified using a method similar to that described in section 5.3 so that the output would appear constant regardless of robot position so long as it remained in view, consistent jittering was observed during robot motion without compensation but was heavily damped with compensation enabled. Beyond this, the effects of latency compensation (or the lack thereof) are hard to quantify because the error introduced by latency is hard to isolate, as it varies heavily with the robot's pose and motion in the environment.

5.3 Planar Obstacle Mapping and Avoidance

As a first step applying vision as a local sensor for navigation, we created flat terrains with colored patches of ground representing "no-step" zones. We then developed the

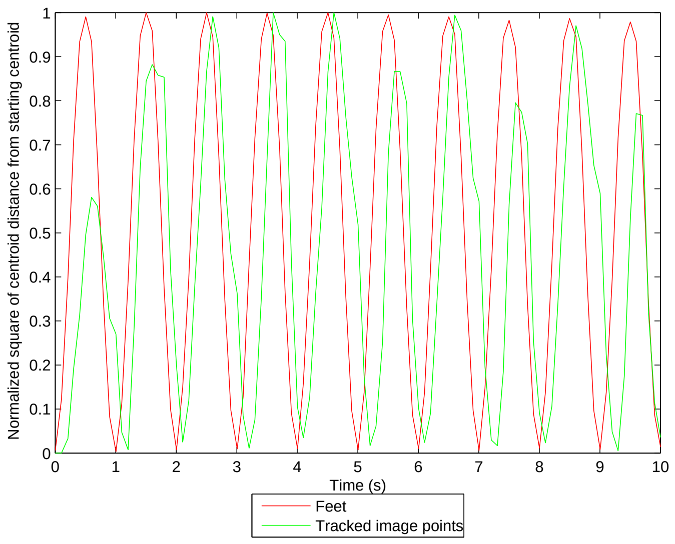

Figure 5.5: Example plot of latency calibration data for USB camera running at 10fps. Here, it exhibits a lag of about 100ms behind the robot state.

necessary control software capable of navigating the robot through such terrains without stepping on these patches. Roughly, this sought to duplicate the results of a similar effort described in section 2.1.3 but by using local sensing rather than a globally-placed camera.

5.3.1 Mapping Planar Obstacles

Before it can navigate around any obstacles, the robot must develop a map containing them. In this case, we developed a mapping procedure that continuously creates a virtual overhead view of the obstacle field as the robot moves.

We start by assuming that when each image frame arrives, an estimate of the camera pose T_{camera_i} with respect to an arbitrarily placed map origin is known. T_{camera_i} may be computed as $T_{body_i} \cdot T_{body \rightarrow camera}$, the latter term of which was calibrated as described in section 5.2.2 and the prior term of which represents the current best estimate of the robot's body position. This estimate in turn may be derived from dead reckoning, some optimal estimation procedure combining that with any other available information (e.g., visual odometry) not completed for this thesis, or (if very high quality results are desired and operation within the motion capture volume is tolerable) from data provided by the motion capture system.

Next, we may note that each point in the image corresponds to a ray in space consisting of all 3-D points whose projection is that image point. The equation for this ray may be found by simply inverting the projection equations given in section 5.2.2:

- $\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} \frac{x-c_x}{f_x} \\ \frac{y-c_y}{f_y} \end{bmatrix}$
- ($\begin{bmatrix} x' \\ y' \end{bmatrix}$ may be derived from $\begin{bmatrix} x'' \\ y'' \end{bmatrix}$ most easily using a numeric equation solver)
- $\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{world} = R_{camera} \begin{bmatrix} tx' \\ ty' \\ t \end{bmatrix} + T_{camera}$, for $t \in \mathbb{R}$

Computing the intersection of the ray corresponding to each corner of the image with the ground plane outlines a quadrilateral patch on the ground, representing (modulo distortion) the area visible in the image. Because the terrain is planar, each point in this area has a projection in the image, and computing its projection to find its corresponding image point gives the ground color (in practice, due to distortion, some points may not have a projection in the image and are disregarded). Repeating this procedure for each point in the ground patch and plotting it in a new image representing an orthographic projection of the ground gives a virtual overhead view of the visible ground area.

To use this to build a map, we can perform this rectification procedure for each incoming frame and overlay them on a virtual overhead image sized to cover the entire ground area of interest. These may be blended to smooth slight inconsistencies, the blending factor for which may be based on the previous pixel's age or some confidence measure (the latter of which was not attempted for this thesis). An example of a map sequence generated by walking forward through a planar obstacle field is shown in figure 5.6.

5.3.2 Navigation Avoiding Planar Obstacles

To navigate an environment containing planar obstacles, we can combine this mapping algorithm with a navigation planner. Conveniently, a high quality footstep planner based on the A^* algorithm written by another member of the Carnegie

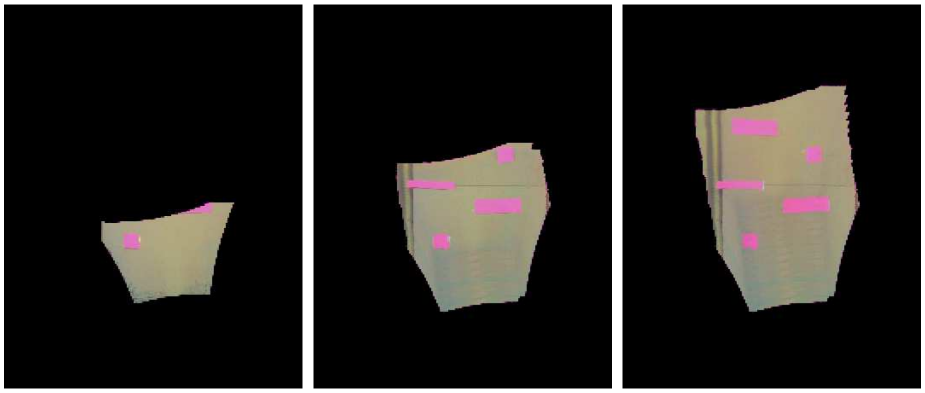

Figure 5.6: Example sequence of map snapshots during planar obstacle mapping of a colored obstacle field.

Mellon Learning Locomotion team was available from that separate effort and was applied for this purpose.

The algorithm for combining the two for navigation purposes may be most concisely described as follows:

- Create an environment map object readable by the footstep planner representing blocked pixels on the ground.
- With this map initially blank, run the planner to find a simple footstep trajectory straight to the desired goal.
- While executing the trajectory, continuously run the mapping algorithm, applying color value thresholding to segment pixel regions corresponding to obstacles.
- Each time the map is updated, retest the remaining footsteps and replan if any land on an obstacle.
- Continue executing the trajectory until the goal is reached or it is determined that no path is available to the goal.

We implemented this algorithm on the robot and demonstrated its successful operation in environments containing a variety of planar obstacle configurations. Several snapshots of one such execution are shown in figure 5.7, along with similar snapshots from its corresponding map with detected obstacles and footstep trajectory.

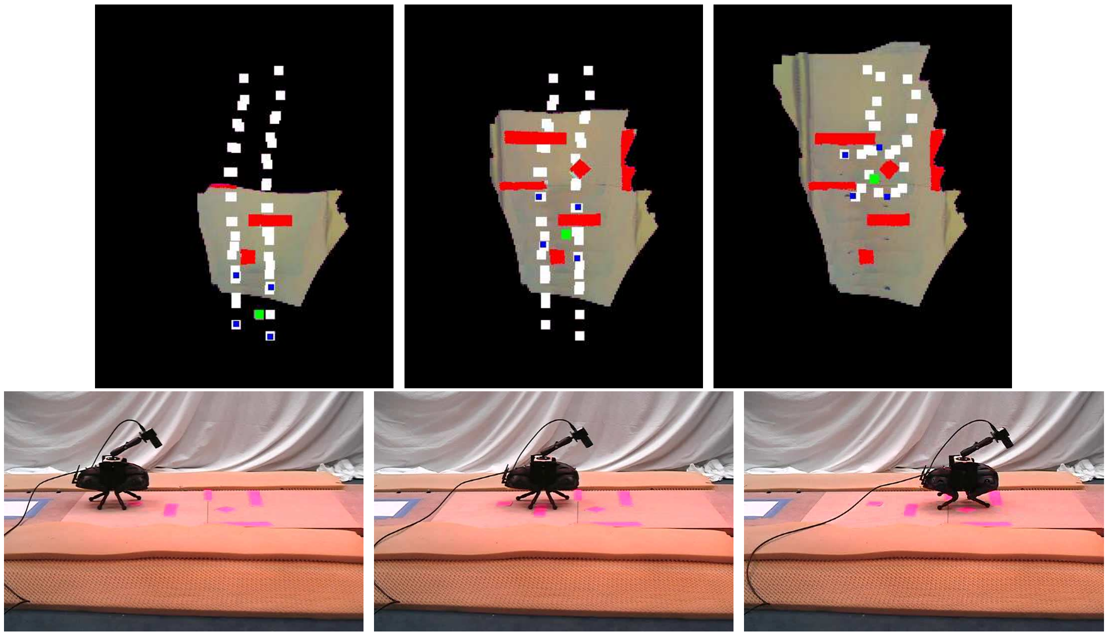

Figure 5.7: Example sequence of map snapshots and live execution during planar obstacle navigation. In the map, red highlights detected obstacles, white represents planned footsteps, blue indicates current foot projections, and green shows the current body projection. Note that between the second and third map images, color variations caused an obstacle to just barely overlap a step placed just adjacent to it, and replanning occurred.

5.4 Three Dimensional Obstacle Mapping and Navigation

The next logical step is of course to navigate fields of 3-D obstacles. Detecting obstacles, perceiving their geometry, and tracking them is an extremely challenging and very processor-intensive task in any generality that remains an active field of research in the computer vision community.

We first noted that it is quite possible to perform sparse 3-D mapping with a single camera by tracking individual points between frames and (given an estimate of the robot's motion between these frames) triangulate to solve for its 3-D location by finding the intersection (or closest point thereto) of the two rays through the two image points. More generally, techniques such as MonoSLAM (Monocular Simultaneous Localization and Mapping) as described in [1] may be applied, giving more complete (but still generally sparse) maps as well as improved robot pose estimates. Other techniques such as the fundamental principles behind stereoscopic vision can be applied between two successive frames and (again given an estimate of the robot's motion) disparity and corresponding depth images formed.

To effectively navigate an obstacle field however, especially with a discrete footstep planner that operates best with contiguous obstacles, we sought a way to easily perceive the complete geometry of obstacles detected in the environment. One of the surest ways to do this is to build a library of geometric models of obstacles that one may place in the environment and instrument physical instances of these with fiducial markers that are easily detectable, trackable, and uniquely identify that obstacle within the library.

A very practical fiducial marking system consists of rectangular markers made of a series of bands of colors such as the one shown in figure 5.8. Though conceptually very simple, detecting and tracking such markers proved somewhat tricky, and as no commonly available software for doing so could be located, we implemented an intuitive algorithm of our own design that proved quite effective.

5.4.1 Fiducial Marker Tracking Algorithm

As initial tests of obvious algorithms for detecting color patches using thresholding demonstrated great weaknesses to even the smallest lighting variations, the algorithm we developed for tracking colored markers representing a painstaking effort to maximize invariance to both lighting conditions and camera color settings such as white balance and saturation parameters.

Given an image believed to contain a marker, we start with a list of the marker colors expected. For our development, only three colors — red, green, and blue — proved necessary to create a well-sized library of markers given that different orderings of these colors could represent different markers. Using this list, we perform a very conservative color thresholding of the image for each color that neglects many points that are of the target color but errs on the safe side by having a much lower chance of highlighting pixels that are not of the target color. Another quick pass is performed to eliminate relatively isolated points (e.g., all highlighted pixels with fewer than 3 highlighted neighbors).

A technique called seeded region growing, inspired by [17] and based upon strategies known to computer vision researchers for years that are essentially the same, each highlighted pixel of a given color in the thresholded image is treated as a "seed" around which a blob of color is grown. Roughly, that algorithm proceeds as follows:

For each color:

1. Put each thresholded pixel (a seed) in a queue and initialize a standard union-find data structure to contain a singleton set (representing a color blob) for each seed.

2. Pop a pixel from the queue, and for each surrounding pixel:

   (a) If the pixel is not part of any blob and is "near enough" to the color of the original seed starting the popped pixel's blob, add it to the union-find data structure in the same set as the popped pixel (e.g., its blob) and push it onto the queue.

   (b) If the pixel is part of a different blob of the same color as the popped pixel, perform a union operation in the union-find data structure between these two blobs (redundant unions are assumed to be harmless).

3. When the queue is empty, terminate.

Upon the completion of this algorithm, it provides a set of blobs containing disjoint contiguous regions of pixels of each color. A minor detail is what is meant by "near enough:" for our purposes, a threshold on the Euclidean distance in RGB space proved sufficient, but fancier metrics possibly utilizing local color variances might be used.

Next, we eliminate blobs that are too small (say, 10 pixels) or too large (say, more than $\frac{1}{k}$ of the entire image, where k is the number of colors expected in the marker to be tracked) to rapidly weed out irrelevant blobs. Then, we select the k blobs nearest in size and proximity to ensure that a set of blobs constituting a roughly contiguous region of the image are chosen, further weeding out irrelevant blobs. To track multiple markers, we may remove these k and repeat the selection process to find the next-best k on which the rest of the algorithm may be repeated.

For each blob, we heuristically detect its edge pixels by highlighting only those with fewer than some threshold of neighbors within the same blob and apply an algorithm utilizing a Hough transform on this edge set to find the lines of the rectangle. Various heuristics are applied to select the best two pairs of lines likely to surround a rectangle, and the lines are intersected to find the rectangle corners. These corners are then refined by performing a local search for the nearby pixel whose image gradient covariance matrix has the largest minimum eigenvalue (the criterion applied in the Sobel corner detection filter).

Finally, the corners are sorted to order the corners of each rectangle in a consistent way and then matched with corners from adjacent blobs to produce a list of corners found on the entire marker, again ordered consistently. An example of a marker being tracked using this algorithm is shown in figure 5.8.

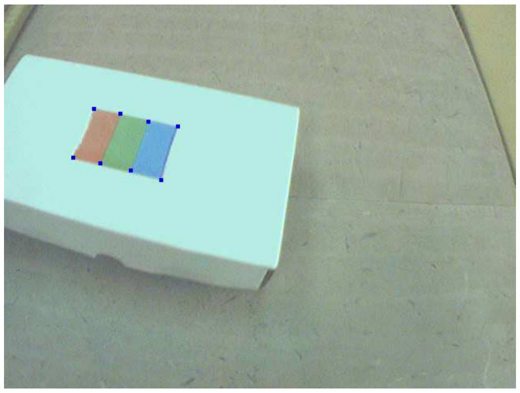

Figure 5.8: Example frame showing detected corners of a tracked colored marker.

This algorithm was implemented with a sharp eye towards performance, and it can detect a single marker approximately 20ms on the desktop-class host computer, which is well below the inter-frame interval of 33ms when operating at 30fps. This may be sped up considerably if this algorithm is only used to initialize tracking and then the corners tracked as features by a Lucas-Kanade point tracker, which can track 10 points in under 8ms. This algorithm would only need to be called infrequently to reinitialize tracking if lost or when the corners no longer appear to frame a coherent marker.

5.4.2 Three Dimensional Tracking and Mapping with Markers

Once a marker has been detected in an image, its known geometry (the sizes of the colored rectangles making it up) gives us a set of point correspondences between the image and the coordinate frame of the marker. With 7 such correspondences (more providing redundancy that increases robustness), we can compute $T_{camera \rightarrow marker}$ using the same calibration function used with chessboard corners in section 5.2.2.

If these markers are affixed to engineered obstacles in a library, the known obstacle models can have associated with them an offset $T_{marker \rightarrow object}$ so that once calibrated against a marker, we can compute $T_{camera \rightarrow object} = T_{camera \rightarrow marker} \cdot T_{marker \rightarrow object}$. Then, assuming that T_{camera} is known as done for planar obstacle tracking, we may derive $T_{object} = T_{camera} \cdot T_{camera \rightarrow object}$.

A world containing such obstacles may then be mapped by using a unique marker for each object, continuously seeking and localizing markers (and thus their attached objects), and updating their position in a 3-D map. When tracking on a marker is lost, the simplest assumption to make is that is that its pose remains the same. Obvious extensions to this (that were not implemented for this thesis) include maintaining degrading pose confidence values or iterating motion (e.g., constant velocity) models for each obstacles. As a very simple example of such mapping in action, figure 5.9 demonstrates localizing a rectangular obstacle in a 3-D map while walking towards it.

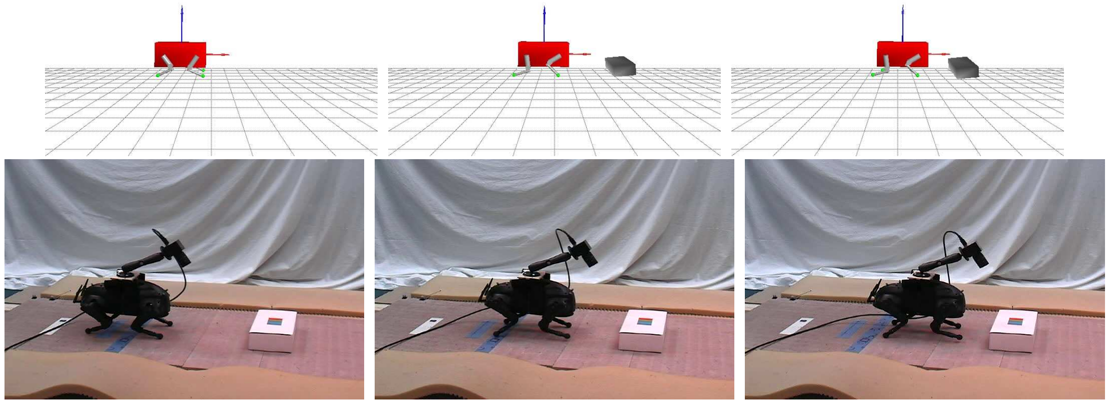

Figure 5.9: Example sequence showing map snapshots and live execution in which a box is tracked in a 3-D world map as the robot walks towards it.

5.4.3 Three Dimensional Obstacle Navigation

Although it was not implemented in the time available, we have reasoned through a simple 3-D analogy to planar obstacle avoidance. We here simply sketch the key structure of an algorithm capable of navigation obstacle fields containing engineered obstacles with fiducial markers corresponding to geometric models in a library:

1. Create an environment map object (this time, in the form of a height map) readable by the footstep planner representing the known 3-D world.

2. With this map initially blank, run the planner to find a simple footstep trajectory straight to the desired goal.

3. While executing the trajectory, continuously run the mapping algorithm described in the previous section.

4. Each time an obstacle appears for the first time or moves significantly, rerun the planner. An improvement to this may include simulating the planned trajectory testing for collisions with obstacles.

5. Continue executing the trajectory until the goal is reached or it is determined that no path to the goal is available.

5.5 Conclusions

In this chapter, we provided a narrative of the process of applying vision as an approach to local sensing. Starting with selecting and attaching a camera to the robot, we stepped through the tasks of preparing it for use via intrinsic, extrinsic, and latency calibration. We then illustrated our successful strategy for mapping and navigation around (or over) planar obstacles. Lastly, we discussed the inherent difficulty of general 3-D tracking, described our implementation of a colored fiducial marker tracking algorithm, demonstrated its application to tracking 3-D obstacles, and then outlined a strategy for extending planar obstacle navigation to 3-D by applying this tracking.

CHAPTER 6

Conclusions and Future Work

In this thesis, we have set out to demonstrate that a great amount of independence in the operation of the LittleDog robot is possible, and we have done just that. We have shown that the robot can be run as a self-contained unit – even outdoors – that is capable of sensing and navigating around its environment. We then described implementations of pose estimation strategies using odometry that rely only on an understanding of the intrinsic kinematic structure of the robot without use of the motion capture system and also without, in the simplest case, any additional sensing. Finally, we recounted our successful effort to use computer vision as a powerful source of local sensing data and exhibited obstacle mapping and navigation in various environments.

At the same time, we stoically acknowledge several fundamental limitations of any effort to grant this robot autonomy. First, the computational power of a desktop-class computer is dreadfully convenient, nay, necessary for any significant image processing required for serious computer vision or real-time motion planning. At present, no reasonably powerful computer is compact enough or has sufficiently manageable power requirements to be practical as an attachable onboard control computer. Somewhat fortunately, this is an artifact of the current state of the art in miniaturization, and it is certain that an amply powerful computer fulfilling these criteria will be available in the reasonably near future. Simultaneously, however, the computational hunger of algorithms will likewise ceaselessly scale, forever perpetuating this conflict.

A second fundamental limitation we must acknowledge is that the extremely high accuracy (sub-millimeter global localization of markers in the best case) provided by the motion capture system is sure to be unmatched by any local sensing techniques, as that system itself is a very expensive, carefully engineered, and globally fixed vision system. Thus, when operation in an enclosed volume and instrumenting the robot (and potentially aspects of its environment) is acceptable, using it is surely wisest. Of course, the much wider "real world" in which we wish this and any other robot to operate extends far beyond its view, and it is then that local sensing becomes critical.

Naturally, the efforts enumerated in this thesis barely begin to scratch the surface of what can be done to enable untethered operation in complex environments. To begin with, a much improved pose tracking strategy could be designed that integrates best estimates from odometry (which itself may be improved by attempting to model backlash and foot slip) with any other information available such as IMU accelerations or that which could be provided by visual odometry or SLAM). This would then improve the fidelity of the vision strategies described in this thesis or any others that require an estimate of the robot's pose.

In terms of immediate extensions to the work depicted in this thesis that could be attempted, the first would be to actually implement the algorithm described in section 5.4.3 for navigation in 3-D terrains of marked engineered obstacles. A logical step from that would then be to explore a 3-D tracking algorithm that does not require fiducials, an example of which might include one that tracks edge features of engineered but unmarked obstacles as described in [8]. From there, we would like to make use of the MonoSLAM algorithm previously mentioned in an effort to implement the improved pose tracking suggested in the previous paragraph.

Thinking further out, one might explore any of the many other alternative local sensors. To start with, other cameras that might be smaller, lighter, wireless, or stereoscopic may enable possibilities for more cameras, offboard processing, or better 3-D mapping. A wide range of range sensors such as an infrared time-of-flight camera (e.g., the Swiss Ranger) or a laser rangefinder (e.g., a Hokuyo or miniature SICK sensor) could be explored.

Additionally, one might give thought to how expected hardware improvements in the next generation LittleDog might be exploited, or, indeed, how the design could be altered arbitrarily to make it easier to achieve autonomy. An obvious start on the latter would be to provide proper attachments for additional sensors rather than forcing ad-hoc approaches such as the saddle we constructed. Better (for instance, multi-axis) foot force sensors (anticipated in the next revision) should help immensely with ground contact estimates, hopefully enabling both greatly improved ground contact detection and perhaps even a better chance at detecting and measuring foot slip, would greatly aid odometry estimation. A better (e.g., higher accuracy with less drift) IMU could likewise aid in pose tracking. Likewise, a tighter gear train or encoders placed closer to the actual joints would lessen the effects of backlash on odometry. Finally, stronger motors would increase the performance and load-bearing capacity of the dog, allowing for the attachment of more sensors and faster execution of motions.

Bibliography

[1] A. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the 2003 IEEE International Conference on Computer Vision (ICCV '03)*, pages 1403–1410, Cote d'Azur, France, 2003.

[2] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 2003.

[3] J.-S. Gutmann, M. Fukuchi, and M. Fujita. A floor and obstacle height map for 3D navigation of a humanoid robot. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA '05)*, pages 1066–1071, Barcelona, Spain, Apr. 2005.

[4] Intel Corporation. OpenCV: Intel Open Source Computer Vision Library [online, cited August 16, 2007]. Available from: `http://www.intel.com/technology/computing/opencv/index.htm`.

[5] S. Kagami, K. Nishiwaki, J. J. Kuffner, K. Okada, M. Inaba, and H. Inoue. Vision-based 2.5D terrain modeling for humanoid locomotion. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA '03)*, pages 2141–2146, Taipei, Taiwan, Sept. 2003.

[6] L. Kavan, S. Collins, C. O'Sullivan, and J. Zara. Dual quaternions for rigid transformation blending. Technical Report TCD-CS-2006-46, Trinity College Dublin, 2006.

[7] A. Lorusso, D. Eggert, and R. Fisher. A comparison of four algorithms for estimating 3-D rigid transformations. In *Proceedings of the 1995 British Machine Vision Conference*, pages 237–246, Birmingham, England, 1995.

[8] P. Michel, J. Chestnutt, S. Kagami, , K. Nishiwaki, J. Kuffner, and T. Kanade. GPU-accelerated real-time 3D tracking for humanoid locomotion and stair climbing. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '07)*, San Diego, CA, Oct. 2007. (Accepted for publication).

[9] P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade. Online environment reconstruction for biped navigation. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA '06)*, pages 3089–3094, Orlando, FL, May 2006.

[10] P. Michel, J. Chestnutt, J. Kuffner, and T. Kanade. Vision-guided humanoid footstep planning for dynamic environments. In *Proceedings of the 2005 IEEE-RAS International Conference on Humanoid Robotics (Humanoids '05)*, pages 13–18, Dec. 2005.

[11] Mobot Committee. Carnegie Mellon Mobot Race [online, cited August 16, 2007]. Available from: `http://www.cs.cmu.edu/~mobot`.

[12] R. Ozawa, Y. Takaoka, Y. Kida, K. Nishiwaki, J. Chestnutt, J. Kuffner, S. Kagami, H. Mizoguch, and H. Inoue. Using visual odometry to create 3D maps for online footstep planning. In *Proceedings of the 2005 IEEE International Conference on Systems, Man, and Cybernetics (SMC '05)*, pages 2643–2648, Waikoloa, HI, Oct. 2005.

[13] K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, and T. Yoshigahara. Obstacle avoidance and path planning for humanoid robots using stereo vision. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA '04)*, pages 592–597, New Orleans, LA, Apr. 2004.

[14] O. Stasse, A. J. Davison, R. Sellaouti, and K. Yokoi. Real-time 3D slam for humanoid robot considering pattern generator information. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, pages 348–355, Beijing, China, Oct. 2006.

[15] Y. Takaoka, Y. Kida, S. Kagami, H. Mizoguchi, , and T. Kanade. 3D map building for a humanoid robot by using visual odometry. In *Proceedings of the*

2004 IEEE International Conference on Systems, Man, and Cybernetics (SMC '04), pages 4444–4449, The Hague, Netherlands, Oct. 2004.

[16] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, Apr. 1991.

[17] Z. Wasik and A. Saffiotti. Robust color segmentation for the RoboCup domain. In *Proceedings of the 2002 IEEE International Conference on Pattern Recognition (ICPR '02)*, pages 651–654, Quebec, Canada, Aug. 2002.

[18] Wikipedia. RANSAC — Wikipedia, the free encyclopedia [online]. 2007. Available from: `http://en.wikipedia.org/w/index.php?title=RANSAC&oldid=148465041`. [Online; accessed 16-August-2007].

[19] Wikipedia. Slerp — Wikipedia, the free encyclopedia [online]. 2007. Available from: `http://en.wikipedia.org/w/index.php?title=Slerp&oldid=121543596`. [Online; accessed 16-August-2007].

[20] H.-J. Yeo, I. H. Suh, B.-J. Yi, and S.-R. Oh. A single closed-loop kinematic chain approach for a hybrid control of two cooperating arms with a passive joint: An application to sawing task. *IEEE Transactions on Robotics and Automation*, 15(1):141–151, Feb. 1999.