Design and Control of the BlueFoot Platform : A Multi-terrain Quadruped Robot

A Thesis
Submitted in Partial Fulfillment
of the Requirements for the
Degree of
Master of Science (Electrical Engineering)
at the Polytechnic University

by

Brian Cairl
May 2015

Master's Examination Committee:
Copy Number: 1

Approved by:

____
Advisor

____
Date

____
Department Head

____
Date

VITA

Mar 23, 1992	Born
Sep, 2010	Entered the NYU Polytechnic School of Engineering as a Mechanical Engineering major and an Honors student.
May, 2012	Entered the Controls/Robotics Research Laboratory as an undergraduate researcher. Began formal design of BlueFoot's predecessor, the GreenFoot Platform.
Jun, 2012	Completed a minor in Mechanical Engineering.
Jan, 2013	Completed undergraduate Senior Design I under the advisement of Professor Farshad Khorrami.
Oct, 2014	Submitted a paper summarizing the design and control of the BlueFoot platform to the 2015 International Conference on Robotics and Automation (ICRA).
Mar, 2015	Submitted a paper on a NARX Neural Network trunk leveling controller for legged robots to the 2015 International Conference on Intelligent Robots and Systems (IROS).
Apr, 2015	Admitted to the NYU Polytechnic School of Engineering as PhD Fellow under the advisement of Professor Farshad Khorrami.

I would like to thank Professor Farshad Khorrami for his advisement throughout the course of this project, as well as for guiding my path of study in the fields of controls and robotics during my career as a BS/MS student. The knowledge and skills I have gained under Professor Khorrami's advisement will be invaluable to my future work and continued learning. I also thank Professor Khorrami the time he spent helping me revise papers and discussing each stage of my project. My time under his advisement has given me a wealth of insight about controls, robotics, and research.

I would also like to thank Dr. Krishnamurthy for his guidance in various implementation matters and providing me with invaluable insight to particular problems that I faced during the course of my work. Additionally, I would like to thank my colleague Griswald Brooks for countless instances of assistance with hardware-related matters during the design and construction of my robot.

I would like to dedicate this work to my loving mother, father, brother and grandparents for thier love and support through all my endeavors.

THESIS ABSTRACT

POLYTECHNIC UNIVERSITY
GRADUATE SCHOOL

NAME: Brian Cairl **QUARTER/YEAR:** Srping 2015

DEPARTMENT: Electrical Engineering **DEGREE:** M.S.

ADVISER'S NAME: Dr. Farshad Khorrami

TITLE OF THESIS: Design and Control of the BlueFoot Platform : A Multi-terrain Quadruped Robot

This thesis presents the development and control of a small-scale quadruped robot platform with 16 actuated degrees-of-freedom, named "BlueFoot." The BlueFoot platform has been developed for the purpose of studying multi-terrain navigation and gait control in concert with full-body actuation, which may be used for reorienting payloads (e.g., laser distance sensor and vision-sensor peripherals). This thesis will detail the design of the BlueFoot platform and its hardware sub-systems; an in-depth analysis of the system's kinematic model and robot dynamics; core Central-Pattern Generator (CPG) based gaiting algorithms introducing reflexive, feedback-driven mechanisms; and a unique foot placement and Zero-Moment Point (ZMP) posture controller based on a virtual force model and a posture feedback loop utilizing inertial measurements.

This thesis offers a method for disturbance rejection and constant orientation of the trunk of a multi-legged (here a quadruped) robot. This is significant when payloads (such as cameras, optical systems, armaments) are carried by the robot. The trunk is stabilized by the utilization of an on-line learning method to actively correct the open-loop gait generated by a central pattern generator (CPG) or a limit-cycle method. The learning method is based on a Nonlinear Autoregressive Neural Network with Exogenous inputs (NARX-NN)– a recurrent neural network architecture typically utilized for modeling nonlinear difference systems. A supervised learning approach is used to train the NARX-NN. The efficacy of the proposed approach is shown in detailed simulation studies of a quadruped robot.

Lastly, this thesis will present several algorithms related to navigation control, terrain modeling, and rough-terrain gait planning. In particular, algorithms for surface reconstruction and foothold planning over uneven terrain will be integral components for future developments related to the BlueFoot project. Results from simulations and actual robot trials will be presented to demonstrate the performance of these control elements.

Adviser's Signature
Department of Electrical
Engineering

TABLE OF CONTENTS

VITA **ii**

LIST OF FIGURES **vii**

LIST OF TABLES **x**

I. Introduction **1**

1.1 Central Pattern Generators for Gait Control 2
1.2 Zero Moment Point Body Placement Control 5
1.3 Trunk stabilization 6
1.4 Potential-Fields Navigation 9
1.5 3D Surface Reconstruction for Rough Terrain Planning 12
1.6 Overview of Thesis 13

II. Hardware and Design **15**

2.1 Overview and Design Goals 15
2.2 Robot Structure 16
2.2.1 Main Body (Trunk) Design 17
2.2.2 Leg Designs 20
2.3 Computational and Sensory Hardware 21
2.3.1 Device Descriptions 22
2.3.2 Device Networking 25
2.4 System Power 26
2.4.1 Power Routing 26
2.4.2 Energy Requirements and Runtime 26

III. Software **28**

3.1 System Software Architecture . . . 28

3.1.1 System Task Allocation . . . 29

3.1.2 Inter-processor Communication . . . 32

3.2 Ground Station . . . 35

IV. System Modeling **37**

4.1 Kinematic Model . . . 37

4.1.1 Forward Position Kinematics . . . 37

4.1.2 Inverse Position Kinematics . . . 41

4.1.3 Velocity Kinematics . . . 43

4.2 Dynamical Model . . . 44

4.2.1 System State Vector and General-Form Dynamics . . . 44

4.2.2 Joint-Servo Dynamics . . . 45

4.3 BlueFoot Simulator . . . 46

V. Gaiting and Gait-Stability Control **50**

5.1 Overview . . . 50

5.2 Central Pattern Generator (CPG) Based Gaiting . . . 50

5.2.1 Reflexive Gait Adaptations . . . 52

5.3 Foot Placement Control . . . 56

5.4 ZMP Based Trunk-Placement Control . . . 59

5.5 Trunk Leveling via NARX-Network Learning Approach . . . 63

5.5.1 NARX-Neural Network . . . 64

5.5.2 NARX-NN Training Regimen . . . 65

5.5.3 Compensator Output . . . 66

5.5.4 NARX-NN Compensator Results . . . 67

VI. Perception and Navigation **73**

6.1 Flatland navigation and goal (feature) tracking . . . 73

6.1.1 LIDAR-based Potential-Fields Algorithm . . . 74

6.1.2 Simulated potential field navigation results . . . 76

6.1.3 Incorporation of Camera-based Feature-Tracking . . . 77

6.2 Towards Rough Terrain Navigation . . . 81

6.2.1 Terrain Mapping from 2D scans . . . 81

6.2.2 Height-map Generation from 3D point cloud 82
6.2.3 Generating a Cost-map from the Height-map 85
6.2.4 Surface Reconstruction 88

VII. Concluding Remarks **92**

A. Leg Jacobian with respect to frame $O_{i,0}$ **95**

LIST OF FIGURES

1 The BlueFoot Quadruped Robot 2
2 Parallel NARX-network model with a linear output layer. 7
3 The BlueFoot Quadruped Robot: single-camera configuration *(left)*; stereo-camera configuration *(right)* 16
4 Root section of trunk. Call-out in top-right shows main-switch access through the bottom of the root module. 17
5 Main section of trunk. Call-out in the top-right shows how the ODROID-XU and AutoPilot computers fit within the module. 18
6 Head section of trunk. Monocular camera configuration with Hokuyo-URG *(left)*; and Stereo configuration with PLDS *(right)*. 19
7 Closeup of BlueFoot's leg. *(left)* shows effective link lengths and the location of defined joint positions. 20
8 Series elastic brackets. 20
9 BlueFoot device networking diagram. 25
10 BlueFoot power routing. 27
11 BlueFoot's on-board processes/signals and their relationships. 30
12 Communication flow between processors on the BlueFoot Platform. 33
13 A visualization of BlueFoot's coordinate frame configuration. 38
14 Range of motion for body's planar position (*left*) and body pitch and roll (*right*) while in the default stance. 43
15 Visualization of the BlueFoot simulator. 46
16 Joint position comparison between the simulated and real quadruped while performing a gait which achieves a forward velocity of $60\frac{mm}{s}$ 48
17 CPG output and phase plots for a two-paced gait over two gait cycles. . 54
18 CPG output and phase plots for a two-paced gait over two gait cycles. . 55
19 CPG output transition during four-pace to two-pace gait switch. 55

20 Real joint angles during a gait at $65\frac{mm}{s}$. 56
21 Virtual foothold representation. Blue arrows represent an attractive "force" between the feet and their corresponding virtual foothold. 58
22 Stepping pattern generated using virtual-force foothold controller during two-pace gait sequence at $v^r = 40\frac{mm}{s}$. 59
23 Stepping pattern generated using virtual-force foothold controller during four-pace gait sequence at $v^r = 40\frac{mm}{s}$. 60
24 Body motion generated using ZMP controller during two-pace *(top)* and four-pace *(bottom)* gaiting sequence. 62
25 Quadruped tipping about planted feet. 64
26 Full system diagram with NARX-NN compensator mechanism. 68
27 NARX Network MSE convergence for trial shown in Figure 32 70
28 Trunk orientation during 60 $\frac{mm}{s}$ gait with $\alpha = 0.125$. 70
29 Trunk orientation during 60 $\frac{mm}{s}$ gait with $\alpha = 0.250$. 71
30 Trunk orientation during 60 $\frac{mm}{s}$ gait with $\alpha = 0.350$. 71
31 Trunk orientation during 80 $\frac{mm}{s}$ gait with mixing parameter set to $\alpha = 0.35$ 72
32 Trunk orientation during 100 $\frac{mm}{s}$ gait with $\alpha = 0.35$ 72
33 Potential field, $f(x)$ with $d_{min} = 0.4m$ given an environment with with obstacles. 75
34 Path (shown in yellow) taken by robot through a simulated set of rooms and halls using the LIDAR-based potential fields navigation scheme. . . . 77
35 Path (shown in yellow) taken by robot through a simulated set of rooms and obstacles. 78
36 Path (shown in yellow) taken by robot through a real-life lab setting showing the robot's view of the goal marker. 80
37 LIDAR swept over a range of angles, $\theta_{b,x} \in [\theta^r_{b,x,1}, \theta^r_{b,x,N}]$. 81
38 Original 3D point-cloud of terrain patch *(top)* and corresponding view from robot's on-board camera *(bottom)*. 83
39 Relative height-map *(left)* and its corresponding gradient *(right)*. 85
40 Height-map showing terrain variation in the $z_{\mathcal{M}}$ direction. 85
41 Projections of a sample cost map generated from the height-map shown in Figure 39. 88

42 Point cloud generated from sequential scans of a room showing flat-surface candidates *(top)* and 3D Point Cloud with normal estimation *(bottom)*. . 91

LIST OF TABLES

1 Link and body-offset lengths for each leg. 21

2 System communication port-pairs and corresponding data transfer rates . 26

3 Power consumption summary by device (for single-camera configuration). 27

4 Battery power supply and estimated runtime summary. 27

5 Structure of the packets sent from Ground-Station to TM4C. 33

6 Structure of the packets sent from the TM4C to the Ground-Station. . . 34

7 Structure of the packets sent from the RM48 to the ODROID. 34

8 Structure of the packets sent from the ODROID-XU to the RM48. 35

9 DH parameters for all legs. 39

10 Imposed joint limits. 42

11 Physical joint limits. 42

12 Locations for the platform and feet when in the default stance, written with respect to the frame $O_{b'}$. 43

CHAPTER I

Introduction

The design of legged robots and associated methods of locomotion control has been an area of interest spanning the past several decades, as shown by [1–5]. Quadruped robotic systems have gained popularity in studies pertaining to variable terrain navigation and full-body stability adaptation. Well known examples of this from the past 15 years are the Tekken [6], Kolt [7], BigDog [8], and HyQ [9] quadrupeds. Many of these systems have been implemented on a larger scale so that they can carry substantial payloads while maintaining adequate system bandwidth for fast gaits and robustness to rough terrains. Few, however, have been implemented on the scale of a hobby-robot platform while still maintaining an aptitude for rough terrain navigation and comparable sensory prowess.

The BlueFoot quadruped is a self-contained, fully-actuated platform with the dexterity to perform stabilization and repositioning maneuvers on variable terrains along the same lines as the LittleDog platform [10]. Namely, BlueFoot has been designed with sixteen actuated degrees of freedom to allow for the execution of a wide range of body and leg articulations. This level of dexterity grants the BlueFoot platform the ability to articulate its trunk over a range of poses, as well as overcome raised or uneven terrain.

BlueFoot is outfitted with several on-board vision sensors, including a LIDAR and camera, which are mounted to its trunk (main body). BlueFoot can articulate (*i.e.*, pitch and yaw) these sensors by reposing its trunk using aggregate leg motion controls. BlueFoot also includes a sizable array of other on-board sensors for feedback and control, including joint position, velocity and loading sensors; an inertial measurement unit (IMU); and foot-contact sensors. Using the computational, sensory and motor capacities at hand, BlueFoot has the ability to utilize similar control mechanisms to those implemented on larger quadruped systems.

The BlueFoot platform inherently demands a variety of control routines to achieve locomotion and system stability, making this robot an ample platform for studies related to gait design and motion planning. In particular, BlueFoot's controllers makes direct use of the system's kinematic model; and involve applications of open-loop gait design and stabilization for the purpose of achieving dynamic locomotion control. In particular, BlueFoot is gaited via a central pattern generator (CPG) based technique which is augmented with a foothold controller along the same lines as [11] and [12]. Active platform stabilization is performed via a zero-moment point (ZMP) based body placement controller which stabilizes the system using planar trunk motions during arbitrary gaiting sequences. Both such controllers make use of virtual-forces to drive system reference commands. These controllers apply BlueFoot's forward kinematic model for the purpose estimating joint and foot positions. Finally, outer-loop control routines are implemented to supply commands and corrections used in system navigation control. Among these controllers are a potential fields navigation controller, which incorporates image-feature tracking; and 3D point-cloud processing routines for surface reconstruction and foot-placement planning.

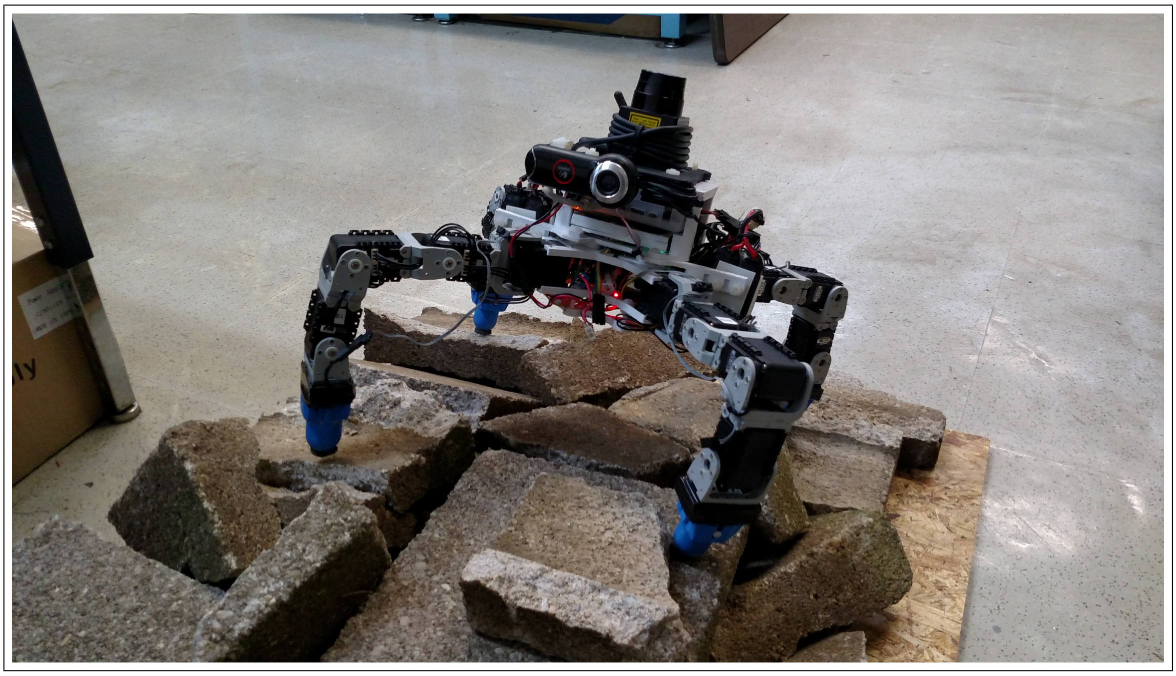


Figure 1: The BlueFoot Quadruped Robot

1.1 Central Pattern Generators for Gait Control

As previously mentioned, BlueFoot's core gaiting routine relies on the utilization of artificial Central Pattern Generators (CPGs). This control mechanism is inspired from biological neural networks which generate rhythmic motions [13]. [14] describes biological CPGs as a form of self-organizing cellular neural network, and also explains the role of limited feedback in CPG networks. In fact, a key feature of these networks is that they can act without explicit sensory feedback inputs or directions from a higher-level command unit, such as a brain. Instead, signals emanating from independent motor units (and, sometimes, feedback gathered from sensory neurons) are utilized to trigger or inhibit a sequence of successive, self-coordinated motor operations. These activation sequences create loops which give rise to cyclic motion patterns.

In robotics, biological CPG's are modeled via an artificial counterpart which applies multi-state unit-oscillators to represent neural units. The dynamics of each unit oscillator in an artificial CPG network are designed to influence the dynamics of other network oscillators through tunable coupling parameters. Typically, the coupled oscillator network is implemented on a digital controller, which numerically integrates the dynamics of each neural oscillator. The output states of each unit-oscillator are used to drive selected robot degrees of freedom. Oscillator outputs could also be used for planning periodic motions in the robot's task space, which are then translated into the joint-space references via an inverse kinematics mapping, as is done in BlueFoot's gait control routine. In implementation, specific motions are achieved through careful tuning of oscillator coupling parameters which incurs particular phase offsets between the individual limit-cycles produced by each unit-oscillator. The ability to coordinate unit-oscillator dynamics is what allows artificial CPGs to be used in the control of higher-level motor tasks, such as walking or crawling.

The selection of a unit-oscillator for use in a CPG network is a fundamental matter in CPG network design. CPG networks can be designed to employ unit-oscillators with varying oscillator dynamics (*i.e.*, varying number of states, tuning parameters, etc.) which exhibit different limit-cycle behaviors. Hopf Oscillators are used in BlueFoot's CPG implementation, as well as in [12, 15, 16]. Other unit-oscillator types which have been applied to CPGs include Van der Poll Oscillators, as detailed in [13], and the Matsuoka Neural Oscillator [17].

Studies dealing specifically the application of CPG's to multi-legged robot gaiting (specifically quadruped, hexapods and octopodal robots) have been carried out by [18–27]. In particular, [13] states that the attractiveness of CPG's in the control of legged robot locomotion lies in the ability to decouple robot motor control, *i.e.*, walking, from higher-level planning, such as navigation and body-posture control. Additionally, CPG's offer an effective way for smoothly switching between gaiting patterns, such as walking, trotting, or pacing, simply through the modification of a few control parameters. Moreover, the application of artificial CPG's greatly reduces the dimensionality of the gaiting control problem for legged robots.

The specific CPG controller implemented on the BlueFoot robot allows for the generation of coordinated motions which can be modified to yield different overall motion patterns without explicitly directing the motion each joint. Moreover, BlueFoots CPG controller does not use a separate unit-oscillator to control the motion of each joint. Instead, four unit-oscillators are used to control the motion of each *foot*. In doing so, CPG outputs are mapped to stepping trajectories in the robot task-space. The resulting task-space reference commands are mapped into angular joint position references using an inverse kinematics solution for the entire robot. This approach gives way to a hybrid gaiting mechanism which combines the conveniences of CPG-based gaiting with the explicitness of strict foot trajectory planning. In BlueFoot's control scheme, foot-placement is prescribed via a separate planning mechanism which is entirely decoupled from the CPG gait controller. This controller hybridization allows a CPG-based motion generator to be applied to gaiting over varying terrains.

Another important aspect of BlueFoot's CPG implementation is the incorporation of feedback mechanisms which modify CPG parameters. The use of feedback towards improving gaiting stability in CPG-based applications was inspired by the work of [6, 17]. Namely, BlueFoot's CPG-based gait generation incorporates inertial feedback signals into its CPG mechanism by using them to modify unit-oscillator amplitudes and modulate unit-oscillator frequencies. Coupling the unit-oscillator dynamics with sensory feedback gives rise *reflexive* motions which can be tuned to help prevent the system from excessive tipping during gaiting. Reflexive motion incorporation, as it is applied into BlueFoot's CPG gaiting scheme, will be covered in more detail in Section 5.2.

Because CPG-based gaits are inherently open-loop motion control routines, a combination of auxiliary mechanisms must be used in concert with the CPG gait controller in order to ensure system stability during gaiting. The incorporation of feedback signals to modify CPG parameters aids in achieving this to some degree, but is usually insufficient for stable walking over largely uneven terrains. Additionally, this method requires very careful parametric tuning to work robustly under a larger variety of terrain conditions. Thus, other means of stabilization have been incorporated into BlueFoot's gaiting routine to aid in stability.

BlueFoot's core stabilization routine applies a concept named *artificial synergy synthesis*. Using this technique, gait control is carried out independently of a stabilization control. Namely, body stabilization is performed by a subset of the robot's degrees of freedom while gaiting is carried out by the remaining [28, 29]. In original implementations of this technique, adaptations to trunk motion were utilized to stabilize the overall motion of the robot utilizing a ZMP-based approach while gaiting is controlled by a fixed-motion routine. Here, body and foot-placement are both controlled as independent, dynamic routines which supply reference commands in the robot task-space.

1.2 Zero Moment Point Body Placement Control

The zero-moment point (ZMP), which is equivalent to the center of pressure (CoP), is formally defined as a point on the ground beneath a walking system at which the net moment acting upon the trunk (referred to as the *tipping moment*) is zero [30]. The concept of ZMP and its application to legged robotics was originally introduced by [28] and expanded upon in [31]. Both such studies apply ZMP theory towards the control of biped robots.

Formally, the ZMP, denoted p_{ZMP}, can be defined using a formulation for the CoP wherein the moments about τ_x and τ_y, the lateral tipping-moments applied to the robot's body in the world frame, are equal to zero. The solution for $p_{ZMP} \in \Re^3$, with respect to a set of N foot contact points $p_{i,e} \in \Re^3$ and N associated applied foot-contact forces $f_{i,e} \in \Re^3$, arises as a bounded set of solutions to the equation

$$\sum_{i=1}^{N} (p_{i,e} - p_{ZMP}) \times f_{i,e} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ * \end{bmatrix} \tag{1.1}$$

where z-coordinate of p_{ZMP}, $[p_{ZMP}]_z$, is strictly zero when the walking surface is flat, as shown in [5]. Here, derivations for flat-surface ZMP motion will be described, but equations associated with this technique can be extended to non-flat terrain motion in a straight-forward way with a conceptually equivalent end result. The following expression is derived from dynamical equations which describe the total angular momentum, $\dot{L}$, about the legged system's center of gravity (COG), p_{COG}:

$$\dot{L} = \sum_{i=1}^{N} p_{i,e} \times f_{i,e} - m_T p_{COG} \times (\ddot{p}_{COG} + \vec{g}) \tag{1.2}$$

where m_T is the total mass of the legged system and $\vec{g}$ is the standard gravity vector. Assuming that all points of foot-contacts exist on a flat plane, *i.e.*, $[p_{i,e}]_z = 0 \ \forall i = \{1, ..., N\}$, and all contact force, $f_{i,e}$ are pointing upward, the p_{ZMP} of the system can be written as

$$p_{ZMP} = \frac{\sum_{i=1}^{N} (p_{i,e} \times f_{i,e})}{\left[\sum_{i=1}^{N} f_{i,e}\right]_z} = \frac{p_{COG} \times (\ddot{p}_{COG} + \vec{g}) + \dot{L} m_T^{-1}}{[\ddot{p}_{COG} + \vec{g}]_z} \in \mathcal{C}_{ZMP} \tag{1.3}$$

where

$$\mathcal{C}_{ZMP} = \text{conv}\left(p_{1,e}, p_{2,e} ..., p_{N,e}\right) \tag{1.4}$$

with conv$(*)$ defining a convex hull generated from a set of input points, $(*)$. $\mathcal{C}_{ZMP}$ is used to represent the solution space of p_{ZMP}. Moreover, $\mathcal{C}_{ZMP}$ places a bound on the angular momentum $\dot{L}$ which results from contact variations presented through $p_{i,e}$ and $f_{i,e}$. Setting $\dot{L} = 0$, a condition is defined for zero tipping. For BlueFoot's ZMP controller formulation, it is also assumed that the acceleration of the COG is sufficiently small, *i.e.*, $\ddot{p}_{COG} \approx 0$ which yields the following, intuitive condition for minimal tipping:

$$\|p_{ZMP} - p_{COG}\| < \epsilon \tag{1.5}$$

where ϵ is a scalar bounding constant used to ensure p_{COG} also falls within the bounded set $\mathcal{C}_{ZMP}$. Thus, the general idea of this ZMP-based controller is to compute an approximate ZMP location and place the center of the robot's trunk (described by the translation p_b) such that the platform's COG approaches its associated ZMP for some arbitrary kinematic configuration. This is done in an effort to reduce $\left\|\dot{L}\right\|$, so as to avoid tipping about the contacting feet.

1.3 Trunk stabilization

In addition to the aforementioned task-space controllers, a learning controller, which features the use of a NARX neural network (NARN-NN), has been studied and evaluated. In essence, this controller learns to approximate disturbance dynamics during periodic gait routines and corrects trunk orientation by administering adaptations to joint position controls. The goal of this NARX-NN based control routine is to achieve a level trunk during locomotion.

A NARX-NN architecture is used in this controller because of its known effectiveness in approximating nonlinear difference systems and making multivariate time-series predictions [32–35]. Moreover, the NARX-NN is a natural fit for a problem of this nature

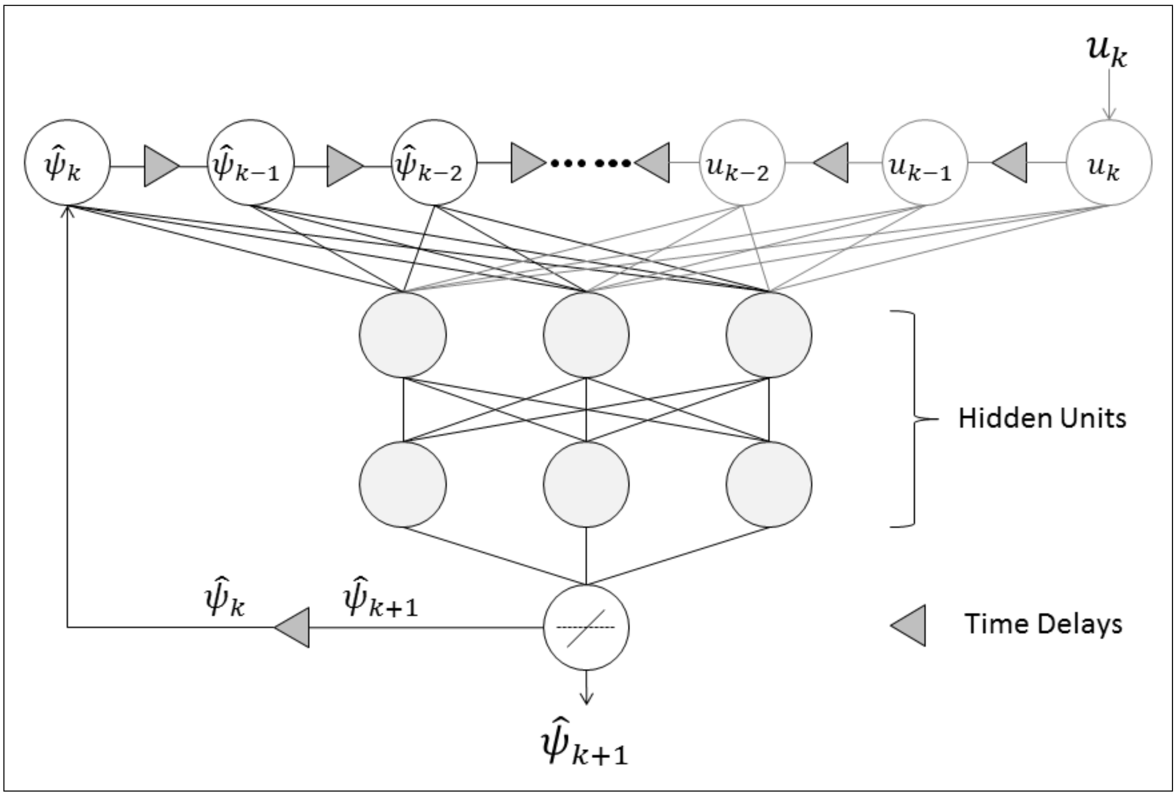


Figure 2: Parallel NARX-network model with a linear output layer.

where the dynamics being considered are both periodic and of a high enough complexity where a nonlinear approximation method is warranted. The parallel NARX-NN model, shown in Figure 2, is comprised of a feed-forward neural network whose input layer accepts a series of time-delayed system state values and network-output histories. The NARX-NN is trained to predict system states in the next time-instant from these inputs. Conveniently, NARX-NN training can be performed using standard BP because recurrence occurs between network inputs and outputs, and not within the hidden layers [36].

An alternative NARX-NN architecture is the series-parallel NARX-NN, in which network prediction target-values are supplied as inputs, as opposed to true network outputs. This formulation is not truly recurrent, but can be trained in the same way as the parallel NARX-NN. As such, this type of network has different convergence characteristics as compared to the parallel NARX-NN. This NARX-NN showed slower convergence than the parallel NARX-NN for the trunk-leveling application to be presented and was not used in the final implementation.

In this controller implementation, the NARX-NN is trained to capture the effects of forces, moments and dynamical couplings that act on the trunk so that an appropriate torque inputs to the joints can be computed. These torque inputs are then used to reduce disturbance effects on trunk orientation while performing the gate. This is achieved by considering the inverse dynamics corresponding to joint motion.

Disturbances imparted upon the trunk during gaiting manifest in the term Φ, largely as a result of variations in f_{ext} and associated effects due to dynamical coupling. Because of this, the NARX-NN will learn an estimate for Φ, denoted $\hat{\Phi}$. The network is trained on-line using the standard incremental back-propagation (BP) algorithm with an adapted learning rate, γ^{lr} and momentum term, μ [37, 38]. This error BP algorithm is a gradient-descent based method used to train a feed-forward neural network with n-layers and layer-connection matrices $\{W^1, W^2, ..., W^{n-1}\} \in W$. The BP algorithm, as used in this control approach, is adapted from [39] and formulated as follows:

$$\Delta W^i \leftarrow -\gamma^{lr}\left(\frac{\partial o^i}{\partial W^i}o^{i-1}\right)^T + \mu\Delta W^i = -\gamma^{lr}\delta^i\left(o^{i-1}\right)^T + \mu\Delta W^i \tag{1.6}$$

where

$$\delta^i = \left(\nabla_y \sigma^i\left(y^i\right)\right)e^i$$
$$y^i = W^i o^{i-1}$$
$$e^i = \left(W^i\right)^T \delta^{i+1}\ \forall\ i \neq n,$$

$\gamma^{lr} \in [0,1]$, the learning and $\mu \in [0,1]$, the learning momentum; $W^i \in \Re^{N_O^i \times N_I^i}$, which represents the weighting matrix between the i^{th} layer (of size N_I^i nodes) and $(i+1)^{th}$ layer (of size $N_O^i = N_I^{i-1}$); ΔW^i, which represents the corresponding weight update to W^i; and e^i is the output error for each i^{th} layer. For the output (n^{th}) layer, e^n is equal to the difference between the network output and the network output target, which will be defined later. For all other layers, e^i represents a *back-propagated* error. from the $(i+1)^{th}$ layer.

$\sigma^i(y^i)$ is an element-wise activation function which outputs a vector of activation outputs, $\sigma_j^i(y_j^i)$ for each j^{th}, weighted input, y_j^i, defined as follows:

$$\left\{\sigma^i(y^i) = \left[\sigma_1^i(y_1^i), ..., \sigma_{N_I^i}^i(y_{N_I^i}^i)\right]^T\ :\ \Re^{N_I^i} \rightarrow \Re^{N_I^i}\right\} \tag{1.7}$$

For the trunk-leveling controller described in Section 5.5, a symmetric sigmoid activation function is used as a hidden-layer activation function. Formally:

$$\sigma_j^i(y_j^i) \equiv \tanh(y_j^i) \in [-1, 1]. \tag{1.8}$$

Hence the gradient which arises for the activation mapping at each hidden layer, $\nabla_y \sigma^i(y^i)$, is defined as follows:

$$\nabla_y \sigma^i(y^i) = \begin{bmatrix} 1-\left(\sigma_1^i(y_1^i)\right)^2 & 0 & \dots & 0 \\ 0 & 1-\left(\sigma_2^i(y_2^i)\right)^2 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & 1-\left(\sigma_{N_I^i}^i(y_{N_I^i}^i)\right)^2 \end{bmatrix} \tag{1.9}$$

given the derivative properties of the $\tanh(*)$ function. For the output layer, a linear activation function is used to avoid output scaling issues. This activation function is defined simply as:

$$\sigma^O(y^O) = y^O \in \Re. \tag{1.10}$$

with $\nabla_y \sigma^O(y^O)$ defined as:

$$\nabla_y \sigma^O(y^O) = I_{N_I^O \times N_I^O}. \tag{1.11}$$

The success of this learning mechanism, as it applies to the trunk-leveling controller to be presented, is predicated on the periodicity of the system dynamics during gaiting. Like any BP-trained neural network, repetition of similar input and output sets is paramount for successful network training and, by extension, prediction accuracy. It is assumed that this specification can be met given the inherently cyclic nature of the dynamics being estimated during gaited locomotion.

1.4 Potential-Fields Navigation

The method selected for navigating the BlueFoot platform over flatland is a potential-fields control approach. This approach is described in [40] for the purpose of controlling robotic manipulators, and analyzed in-depth in [41]. In particular [41] presents shortcomings of this approach, which will be covered in-brief later in this section. Despite its pitfalls, potential fields navigation methods offers a relatively simple and intuitive approach to robot navigation and fits well into mobile robotic tasks which involve "wandering"-type navigation over flat-regions. In particular, this approach is applied to situations where the robot has yet to acquire any knowledge of its surroundings. In the approach to be presented, potential fields navigation is coupled with a camera-based feature tracking, which is used to guide the robot towards features of interest within the unknown environment.

The potential fields approach is used in mobile robot navigation by moving the robot according to a guiding virtual force-vector, F_{nav} [41,42]. This vector is comprised of a sum of virtual repulsive forces, F^- (typically generated from range-sensor data), and virtual attractive forces, F^+, which pull the robot towards known goals. Moreover, F_{nav} formally defined as:

$$F_{nav} = F^+ + F^- \tag{1.12}$$

The general form for the force components F^+ and F^- (represented as F_c in equation (1.13)) is as follows:

$$\begin{aligned} d_k &= p_{POI,k} - p_{robot} \\ F_c &= \alpha_F \sum_k \left(f(\|d_k\|) \frac{d_k}{\|d_k\|} \right) \end{aligned} \tag{1.13}$$

where p_{poi} and p_{robot} represent the position of each k^{th} point-of-interest (POI) and the position of the robot platform; $f(*)$ is a potential function which returns a scalar potential factor with respect to a scalar argument $(*)$; and α_F is a scaling parameter which is positive when the POIs considered represent goals and negative when POIs represent obstacles to avoid. The potential function, $f(*)$, and scaling factor, α_F, are designable for particular applications. For BlueFoot's navigation scheme, attractive and repulsive forces are generated using a consolidated, piecewise forcing function which is used to guide through an environment where a goal is not specified before hand.

According to [41] the main pitfall with potential fields navigation is susceptivity to local minima within the global force-field. At a local minimum, the F^+ and F^- are of nearly equal magnitude, causing the magnitude of the total guiding force vector, F_c, to be close to zero. In practice, reaching a point at which robot will be completely stationary is unlikely, as sensor readings used to observe environmental obstructions are corrupted by noise. In turn, this noise induces random perturbations in the robot's motion. In fact, perturbations due to sensor noise could actually aid in relieving a situation in which a robot is stuck in a local minima. However, the gradient of the force-field around a minimum point could be very steep over a large area around the aforementioned singularity. These type of potential-sink regions could cause the robot to exhibit limit-cycle behavior as it periodically overshoots and re-attracts to the location of the force-minimum.

This can be overcome, in part, by adding an artificial *inertia* (essentially a tunable gain parameter) when updating the robot's navigation reference signals. Given a set of robot navigation command parameters, v^r and ω^r, and potential reference outputs, v^r_L and ω^r_L (which are generated from F_{nav}), navigation updates with an added update-inertia, $B^r \in \Re^{2\times 2}$, exhibit the follow controller dynamics:

$$\begin{bmatrix} \dot{v}^r \\ \dot{\omega}^r \end{bmatrix} = B^r \left(\begin{bmatrix} v_L^r \\ \omega_L^r \end{bmatrix} - \begin{bmatrix} v^r \\ \omega^r \end{bmatrix} \right) \tag{1.14}$$

where B^r is a strictly positive definite, diagonal "inertia" matrix. This control formulation is equivalent to an outer-loop P-control scheme. Using this update scheme may cause the robot platform to sufficiently overshoot minimum points such that it leaves the local attraction field. Care must be taken in the selection of B, however, so that system still exhibits stable behavior during navigation control. A more sophisticated approach to escaping local minima is mentioned in [43], which involves a "stuck" detection algorithm. The idea behind such an algorithm involves a deduction about whether or not the robot is captured in a local minima based on samples of the robots motion state (*i.e.*, position and velocity). Once the robot has determined that is it stuck, it is instructed to execute a small random-walk as a means of escape.

Here, the local minima problem is addressed through the use of auxiliary (possibly dynamic) target points, as done in [42] via a hybrid navigation potential-fields / visual-servoing scheme. Instead of incorporating goal points directly into the potential fields scheme, goals (in this case, image features) are tracked using an entirely separate navigation scheme. Here, a separate set of navigation reference commands, v_C^r and ω_C^r, are mixed with commands generated via the potential fields algorithm. The amount of influence either command scheme has over the final navigation command parameters, v^r and ω^r, depends on relative measure of "closeness" to the object being tracked, which the robot determines in an image processing routine. The specifics of this will be described in VI.

Hence, the potential-fields portion of this control scheme is used only for the purpose of avoiding potential obstructions, sensed via LIDAR range data. Image-features are used in a visual-servoing routine which guides the robot toward features-of-interest which fall within the robots camera gaze. This approach offers the ability to manually guide the robot during navigation, by either a human overseer, or a partner robot (which could wear trackable markers), as it performs an independent obstacle avoidance routine. The advantage of this approach lives in its simplicity, as it relies only on immediate environmental samples and, thus, has relatively minimal implementation demands. As a mechanism for partially-guided wandering-type (random) navigation within an unknown region, this approach is certainly adequate as a will be shown via empirical results from real-world trials.

1.5 3D Surface Reconstruction for Rough Terrain Planning

The previously introduced navigation scheme is utilized for flatland navigation, exclusively. As such, it is generally insufficient for navigation and planning over rough terrain. Navigation and footstep planning over rough terrain require the acquisition of 3D surface data from the robots immediate environment, generally with high feature detail. This thesis will provide the preliminaries for rough-terrain navigation by way of several surface reconstruction / representation methods from point-cloud data. First, a method for composing 3D point clouds from successive 2D LIDAR scans will be described. Then, algorithms for generating 3D height-maps will be described and a method for cost assignment based on generated height-maps will be offered for use in rough terrain foot-placement planning. Finally, surface reconstruction from raw point-cloud data will be described.

Work related to height-map generation and associated cost-assignment (used in generating discrete cost-maps) from 3D point cloud data is born from an intuitive evaluation of the problem of rough-terrain planning. This work should be viewed as the initial steps towards formal rough terrain planning which will later be merged with existing research which relate to this method of terrain representation. On the other hand, implementations related to 3D surface reconstruction from 3D point clouds rely heavily on algorithms originally outlined in [44] and formally implemented in the OpenPCL library [45].

Here, surface estimation from 3D point cloud data is approached by way of successive plane-fitting (normal estimation) via Principle Component Analysis (PCA) on a moving subset of points [44, 46]. Specifically, a normal vector is associated with each point, $\bar{x}_i$, within the point cloud, $\bar{\mathcal{S}}$, by fitting a plane to a subset of neighboring points which fall within a unit-ball (of radius d_s) around $\bar{x}_i$. Here, $\bar{\mathcal{S}}$ represents a raw 3D point cloud. This point cloud is pre-conditioned to generate a sparser point cloud, $\hat{\mathcal{S}}$, which estimates the collection of points $\bar{\mathcal{S}}$. Preconditioning involves a voxel-grid filter, which is used to down-sample the original point cloud; and a moving-least-squares (MLS) filter, which is used to regularize points and remove outliers. These preconditioning routines aid in the normal estimation process by decreasing computational burden of the estimation algorithm. This is achieved by reducing the density of the space which must be searched for nearest neighbor points about each $\bar{x}_i$. Preconditioning also improves the relative smoothness of the reconstructed surface [44].

To estimate a local plane (via PCA) about the point $\bar{x}_i$, we define a moving-neighborhood, $\mathcal{B}_{\bar{x}_i}$, as follows:

$$\{\bar{x}_j \in \mathcal{B}_{\bar{x}_i} \ : \ \|\bar{x}_j - \bar{x}_i\| < d_s \ , \ \forall \ i \neq j \ \} \ , \ \mathcal{B}_{\bar{x}_i} \subset \bar{S} \tag{1.15}$$

Additionally, the covariance matrix, $C_{\bar{x}_i} \in \Re^{3\times 3}$, of the subset $\mathcal{B}_{\bar{x}_i}$ is defined as:

$$C_{\bar{x}_i} = \frac{1}{S(\mathcal{B}_{\bar{x}_i})} \sum_{\bar{x}_j \in \mathcal{B}_{\bar{x}_i}} (\bar{x}_j - \bar{x}_{c,i})(\bar{x}_j - \bar{x}_{c,i})^T \tag{1.16}$$

where the centroid, $\bar{x}_{c,i} \in \Re^3$, of the subspace $\mathcal{B}_{\bar{x}_i}$ is defined as:

$$\bar{x}_{c,i} = \frac{1}{S(\mathcal{B}_{\bar{x}_i})} \sum_{\bar{x}_j \in \mathcal{B}_{\bar{x}_i}} \bar{x}_j \tag{1.17}$$

and $S(\mathcal{B}_{\bar{x}_i})$ defines the number of points within the subspace $\mathcal{B}_{\bar{x}_i}$. Note that $C_{\bar{x}_i}$ is a positive semi-definite matrix with all eigenvalues, $\lambda_{x_i,j} \geq 0 \ \forall \ j \in \{1,2,3\}$ such that $\lambda_{x_i,0} \leq \lambda_{x_i,1} \leq \lambda_{x_i,2}$. The unit-eigenvectors $v_{x_i,j} \ \forall \ j \in \{1,2,3\}$ represent the principal components of the local subspace $\mathcal{B}_{\bar{x}_i}$, and are defined by:

$$\left\{v_{x_i,j} \ \Re^3 \ : \ C_{\bar{x}_i} v_{x_i,j} = \lambda_{x_i,j} v_{x_i,j} \ \forall \ j \in \{1,2,3\}\right\} \tag{1.18}$$

The eigenvector $v_{x_i,1}$ with corresponding smallest eigenvalue $\lambda_{x_i,1}$ represents a unit-normal which emanates from a 2D manifold of $\mathcal{B}_{\bar{x}_i}$ whose origin is located x_i. This manifold represents a planar least-squares best-fit for the information contained in $\mathcal{B}_{\bar{x}_i}$. To fortify this definition, we can also consider that the minimum eigenvalue $\lambda_{x_i,1}$ $v_{x_i,1}$ indicates that $v_{x_i,1}$ is a direction in $\mathcal{B}_{\bar{x}_i}$ with the lowest variational *energy*. Hence, the sum of the square-errors between all points in $\mathcal{B}_{\bar{x}_i}$ is minimized in the $v_{x_i,1}$ direction.

Given a preprocessed point cloud $\hat{\mathcal{S}}$ (generated from a raw point cloud $\bar{\mathcal{S}}$), the full normal-estimation algorithm is then implemented using the PCA formulation defined from (1.15)-(1.18) in Algorithm 1. Here, $\vec{n}_i \in \hat{N}_{\mathcal{S}}$ represents a normal emanating from each i^{th} point, $\hat{x}_i \in \hat{\mathcal{S}}$.

1.6 Overview of Thesis

This thesis will first detail the major hardware components; design considerations; and construction of the BlueFoot platform. Next, BlueFoot's software and processing architecture, applied in the control the BlueFoot platform, will be described. Thereafter, the kinematic and dynamical models of the BlueFoot system will be described, followed

Algorithm 1 Surface-normal estimation from a preprocessed 3D point cloud.

init: $\hat{\mathcal{S}}, \hat{N}_{\mathcal{S}} = \phi$

for all $\hat{x}_i \in \hat{\mathcal{S}}$ **do**

$\mathcal{B}_{\bar{x}_i} = \text{getNearestNeighbors}\left(\hat{\mathcal{S}}, \hat{x}_i, d_s\right)$

$\vec{n}_i = \text{performPCA}\left(\mathcal{B}_{\bar{x}_i}\right)$

$\hat{N}_{\mathcal{S}} \leftarrow \hat{N}_{\mathcal{S}} \bigcup \vec{n}_i$

end for

by control routines which are presently implemented to gait, stabilize, and navigate the BlueFoot platform. The final section of this thesis will contain concluding statements about the system's design and control, including remarks about possible future directions of study related to the BlueFoot platform and legged robotics as extensions of the existing work.

CHAPTER II

Hardware and Design

2.1 Overview and Design Goals

The BlueFoot quadruped robot is designed as a small-scale, general-purpose legged mobile platform with enough physical dexterity and on-board computational/sensory power to perform complex tasks in variable environments. BlueFoot's hardware configuration is aimed at performing of tasks as a standalone unit, i.e. without power tethering or off-board processing. BlueFoot's sensory, computational and power-source outfit make it fit to complete tasks in both settings fully and semi-autonomous modes.

The implementation of a legged robot which meets these general specification is inherently bottlenecked by several well-known shortcomings which plague legged robot design. These drawbacks can be summarized as follows: relatively low payload capacity, as leg joints are often subjected to substantial dynamic torque loading during gaiting; and higher power consumption due to a, typically, larger number of total actuators. Thus, a general-purpose, multi-legged system like the BlueFoot platform must ultimately achieve a balance between payload-carrying capacity (i.e. maximum joint-servo output torque); actual on-board payload; and on-board energy supply. It is desirable that the sensory and computational power; as well as overall mobility of a legged system are simultaneously maximized along with the aforementioned characteristics.

The design goals which have guided the implementation of the BlueFoot quadruped have been tailored to a yield an overall system design which is both feature rich, computationally powerful, and exploits the natural dexterity and terrain handling of legged robotic systems. Namely, the core design requirements which have guided BlueFoot's development can be summarized as follows:

- The use of legs with joint redundancy for improved dexterity
- The use of smart servos for extended joint feedback and control
- A distributed on-board and computing architecture for hierarchal task handling

- A vision sensor array including a camera and laser-ranging sensors
- 30+ minutes of total battery life

This chapter will outline how an implementation meeting these design goals is achieved, starting with the structural layout of the system. Next, major system payloads and the associated interfacing of major devices will be described. This section which will include details about BlueFoot's actuators, computational modules, and sensory mechanisms. Lastly, the system power routine, energy requirements, and runtime will be detailed.

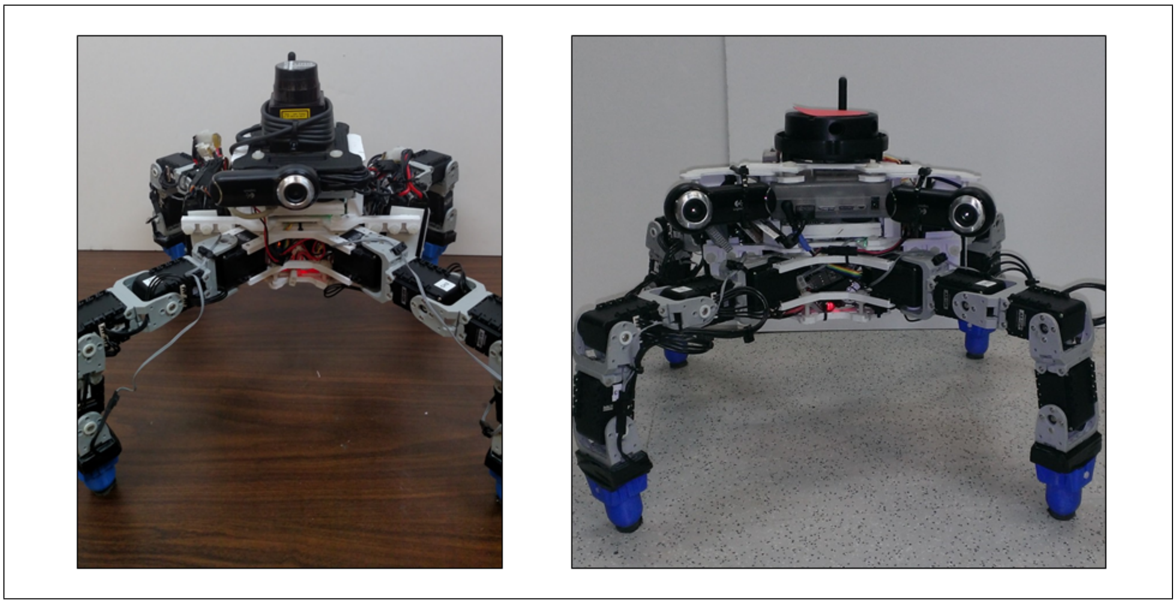


Figure 3: The BlueFoot Quadruped Robot: single-camera configuration *(left)*; stereo-camera configuration *(right)*

2.2 Robot Structure

BlueFoot's body is designed in a modular fashion and is comprised of mostly custom designed, 3D printed parts. The use of 3D printing as a fabrication method allowed for rapid design iterations the early stages of system prototyping, and has kept the weight of the robot's overall structure relatively low. Parts were mainly printed from both PLA and SLA plastics. BlueFoot's overall weight (when fully outfitted) is 1.85 – 1.98 kg, depending on configuration.

The modularity of BlueFoot's overall structure arises from the inherent design requirements associated with 3D printing and general design practices aimed at keeping the system reconfigurable for the incorporation of updated sensory and computational hardware. Moreover, parts are designed to fit future replacements while conforming to the constraints imposed by the 3D printing fabrication method, i.e. particular part size and orientation requirements. Such constraints had to be met by each designed part to ensure print feasibility.

The BlueFoot platform has undergone several minor redesign phases since its inception. These redesigns were necessary to bring the BlueFoot platform to its final structural and hardware state and were performed to accommodate changes in sensory/computational hardware. The sections that follow will mainly focus BlueFoot's final hardware configurations.

2.2.1 Main Body (Trunk) Design

BlueFoot's trunk consists of the three main sections: a lower module which interfaces the legs with the main body; a center chassis, designed to hold computational and battery payloads; and a top platform, which interfaces the system's visions sensors to the trunk. These sections will be referred to as the *Root* module, the *Main* module, and the *Head*module, respectively. The full trunk (not including sensor dimensions) fits within a 21.6 by 21.6 by 15.3 cm bounding box.

The Root Module

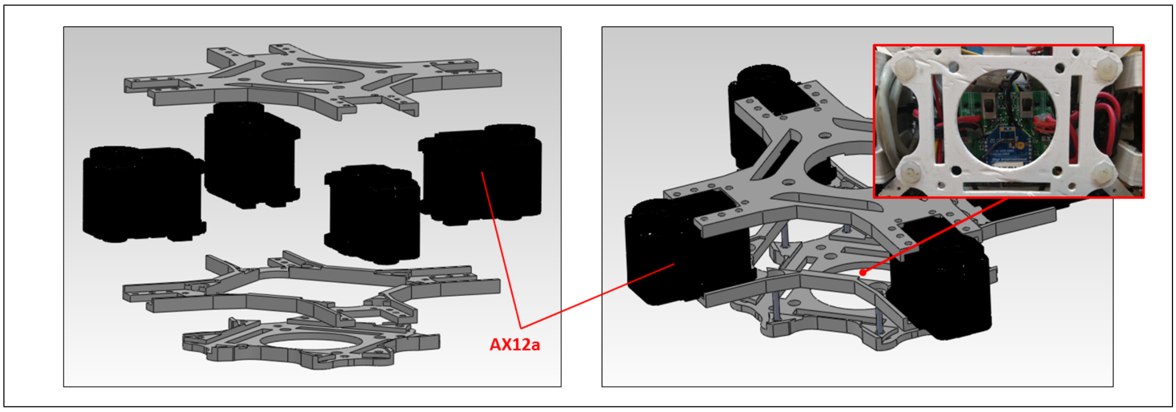


Figure 4: Root section of trunk. Call-out in top-right shows main-switch access through the bottom of the root module.

The Root module, consists of three plates, as shown in Figure 4. Each plate is designed with a central opening to allow for wired connections to pass to other trunk modules. Two such plates directly interface with four servos, which are mounted to four symmetric arms which extend from the center of each plate. These servos are the first joint (hip-joint) of each leg. Each servo mounts to the top a bottom plates via mounting holes located at the top and underside of each servo chassis. The assembly is mated with small steel bolts. A third, smaller plate is attached to the bottom of the module to provide more space for power components and associated wiring. This plate is attached to the bottom of the module via plastic standoffs. An opening in the middle of this plate provides access to the system's main power switches, as well as a removable XBEE wireless radio unit.

The Main Module

The Main module of BlueFoot's trunk includes compartments for an in-house designed AutoPilot unit and a main computer unit, an ODROID-XU. The Main module is designed such that the AutoPilot and ODROID-XU computer slide in and out of the body. The computer payloads are locked into position when the Head module is added to the assembly. The Main body section is designed to fit both computers when stacked upon one another, as depicted in Figure 5. The computer stack is positioned directly in the center of the module when inserted.

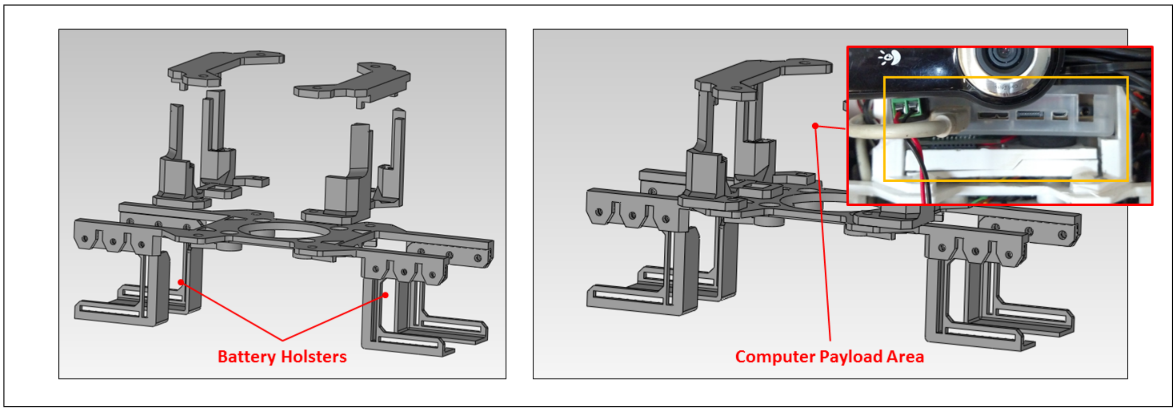


Figure 5: Main section of trunk. Call-out in the top-right shows how the ODROID-XU and AutoPilot computers fit within the module.

The Main module also includes two battery holsters, which hang over its left and right sides. The holsters align the battery packs with the center of Root module. This battery placement serves to lower the center of mass (COM) of the trunk. Doing so serves to lower the magnitude of dynamic torques imparted upon the leg servos during gaiting by decreasing the net moment due to gravity imparted upon the system when the body is oriented away from the direction of gravitational force. The entirety of the Main module is attached to the Root module through the battery holster sub-assembly by four plastic bolts.

The Head Module

Two separate Head modules have been designed for the BlueFoot system : one of which features a stereo camera pair and a Piccolo LIDAR sensor (PLDS); and a monocular design, which features a camera and a Hokuyo-URG LIDAR sensor, as shown in Figure 6. Each head module is attached to Main module via four plastic mounting screws. In the stereo-camera design, two adjustable wings are attached to either side of a top platform which hold cameras. These wings were designed to be adjustable to aid in stereo-camera configuration and calibration. The position of each camera on the trunk allows for a persistent field of view by each camera during mobilization. The PLDS unit is positioned such that the center of its rotating laser head is aligned to the center of the trunk.

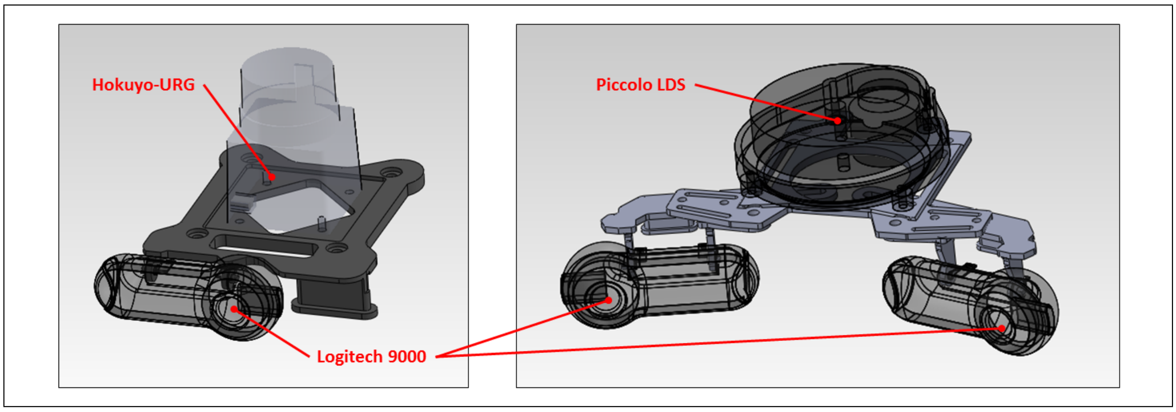


Figure 6: Head section of trunk. Monocular camera configuration with Hokuyo-URG *(left)*; and Stereo configuration with PLDS *(right)*.

In the monocular design, a single camera is mounted such that the lens of the camera is aligned to the sagittal plane of the trunk. This configuration is currently be used as BlueFoot's *primary* head configuration and is mainly being used for 3D point-cloud building and surface reconstruction via 2D LIDAR scans. This is because the Hokuyo-URG laser scanner used in this configuration offers higher-resolution laser-scan outputs, which will be covered in more detail later in this chapter.

2.2.2 Leg Designs

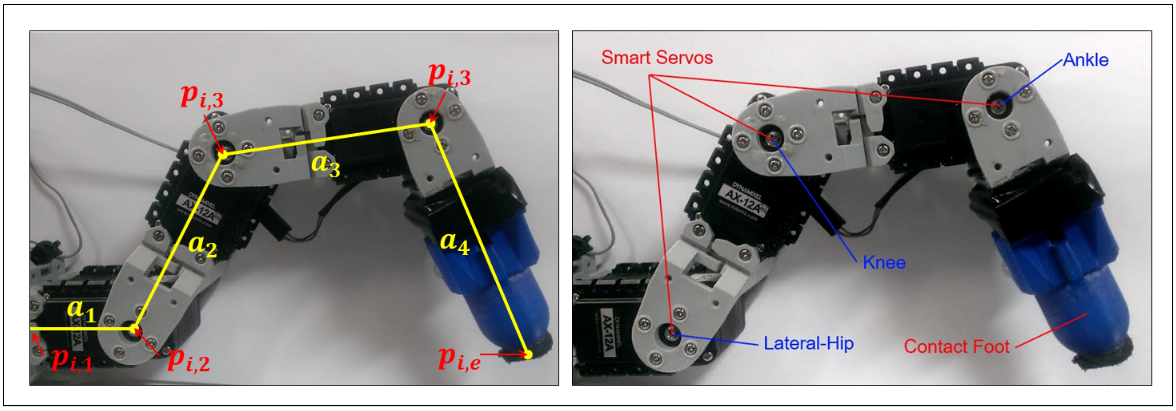


Figure 7: Closeup of BlueFoot's leg. *(left)* shows effective link lengths and the location of defined joint positions.

Each of BlueFoot's legs are identical and are comprised of four Dynamixel AX12a smart-servo actuators (see Figure 7). These actuators are connected via dedicated Dynamixel mounting brackets. Feet are attached to the ends of each leg which contain an embedded, two-state contact sensors. Each foot is designed with a spherical tip, which is rubberized to provide extra grip. The ankle joint of the platform has been added such that the platform can reconfigure its foot orientation while retaining a constant spatial position during gaiting. Additionally, this configuration allows for a considerable amount of independent body re-orientation and repositioning. This capability extends itself to the stabilization and gimbaling of vision sensors mounted on the upper body of the platform while the platform is in motion.

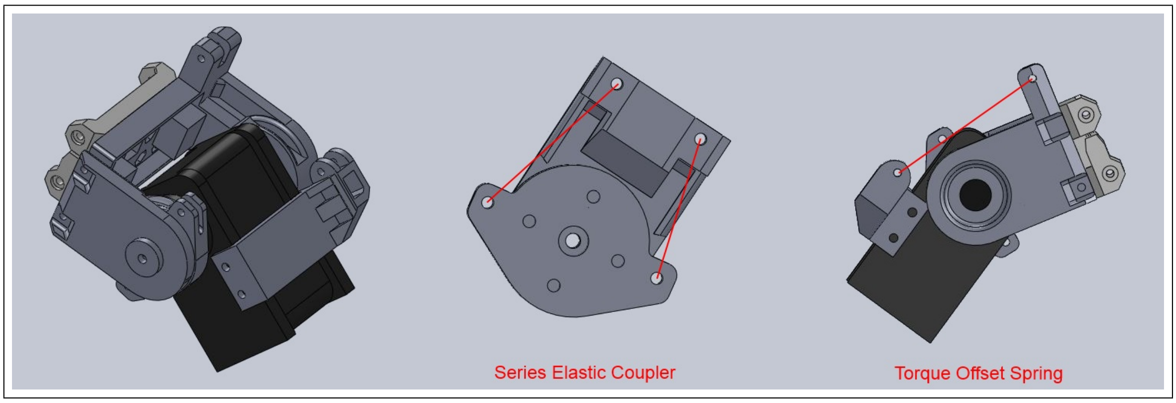


Figure 8: Series elastic brackets.

Though not kept in the system's final design, some experimentation was performed with the incorporation of series elastic joints, which were designed to relieve joint impact during gaiting. Series elastic actuation was achieved by replacing bracket interfacing the first and second hip joints of each leg with an elastic-compliant mounting bracket. This bracket includes spring loaded member which was mounted to the horn of the second hip servo on each leg, as shown in Figure 8 and allowed the leg to deflect a small amount at the lateral hip (second joint).

Link lengths, a_1, a_2, a_3 and a_4; and offset from the center for the Root module to the first joint of each leg, ν are defined in Table 1. These parameters are identical for each leg, and are corresponded with physical leg members labeled in Figure 7.

Link	**Length, m**
a_1	0.06500
a_2	0.06500
a_3	0.06500
a_4	0.06500
ν	0.09215

Table 1: Link and body-offset lengths for each leg.

2.3 Computational and Sensory Hardware

Major payloads on-board the BlueFoot robot are as follows:

- Dual processor AutoPilot unit with a 12-axis inertial measurement unit
- ODROID-XU Computer
- Logitech 9000 Web-cameras
- Hokuyo-URG / Piccolo LIDAR Units (configuration dependent)
- Two-state foot contact sensors (x4)
- Dynamixel AX12a Smart Serial Servos (x16)
- XBEE Wireless radio

Device selection has remained mostly consistent since the platform's inception and initial design, with the exception of computing its main computing units. The AutoPilot unit was updated from an older model, and the ODROID-XU computer replaced a Beaglebone computer for the sake of improving overall computing power.

2.3.1 Device Descriptions

AutoPilot

A dual processor AutoPilot unit performs BlueFoot's low-level gaiting and actuator control tasks, as well as handles communications with a computer running ground-station software. Given the set of low-level sensory and motor-handling tasks it performs, this module has been named the the "Lower Brain" (LB) of the system. The AutoPilot consists of two processing units: a TM4C and RM48 micro-controller (MCU), which operate at 80 MHz and 220 MHz, respectively. These processors communicate over a single UART line, which is used to transfer packeted data between the two processors using a unified inter-processor data transfer protocol, EXI. This protocol which will be described later in more detail. One UART of the RM48 MCU is also connected to an on-board computer, an ODROID-XU, through a USB-to-serial connection. The AutoPilot is powered via an external 12 V supply.

This AutoPilot unit includes a 12-axis inertial measurement unit (IMU) which consists of two, 3-axis accelerometers; one 3-axis rate gyro; and a 3-axis magnetometer unit. This sensor is used for acquiring angular rate data of BlueFoot's trunk and estimating of trunk orientation states using an Extended Kalman Filter (EKF).

ODROID-XU

An ODROID-XU performs many of system's high-level planning tasks, such as navigation, image processing and terrain reconstruction; and handles data data acquisition from both camera and LIDAR sensor units. Given that this unit performs mostly high-level planning tasks, it has been given the name "Upper Brain" (UB). This computer contains a 1.6 GHz, quad-core processor with 2 Gb of RAM. The ODROID-XU can be communicated with over WiFi via a USB WiFi antenna. Currently, SSH tunneling is used to start processes on the ODROID remotely and stream data. The ODROID-XU is powered via an external 5V connection.

Logitech 9000 Web Cameras

Logitech 9000 web cameras have been selected for creating a stereo camera pair, as well as for use in a single camera configuration. These cameras are high-definition web cameras and have a maximum frame rate of 30 fps and a max resolution of 1280 by 720. Cameras are currently read at a the max rate of 30 fps at a more conservative resolution of 640 by 480. These settings are adequate for image processing tasks and have been chosen to reduce nominal data throughput. These cameras are interfaced with the UB (OROID-XU) over a USB connection.

Laser Distance Sensors (PLDS and Hokuyo-URG)

The Piccolo Laser distance sensor (PLDS), which is used in BlueFoot's stereo-camera type configuration, is a 4 meter spinning-head laser range finder. The PLDS has a resolution of a point per degree and covers a range of 360 degrees. Ranging frames (which covers a full rotation) are acquired at a rate of 5 Hz, and are dispatched over a serial connection at 115200 baud. An FTDI break-out board is used to convert the sensor's raw serial output to USB protocol so that the sensor can be interfaced with the UB unit. The PLDS is powered via and external power 5 V source, which is regulated to a 3.3 V voltage level for powering the motor which spins the laser head, and 1.8 V for internal logic. Regulation is performed by an auxiliary power circuit.

The Hokuyo-URG Laser Distance sensor, which is used in BlueFoot's single-camera head configuration, has a a range of 5.6 metets and an angular resolution of 0.38 degrees per point (628 points per scan). This scanner covers a total angular range of 240 degrees. Ranging frames are acquired at a rate of approximately 10 Hz and dispatched directly over a USB connection at 115200 baud. The unit is powered directly over USB.

Foot Contact Sensors

Binary-state contact sensors are embedded in each foot. These contact sensors are essentially limit-switches which generate an active-low signal when the foot comes in contact with the ground. Each sensors is connected to ground and a GPIO pin on the TM4C MCU of the AutoPilot. A 500 Ω is added in series with the limit-switch for the purpose of pin protection.

Dynamixel AX12a Smart Serial Servos

BlueFoot uses 16 Dynamixel AX12a servo units (4 per leg). These servos are position-controlled and commanded over a daisy-chained, half-duplex serial bus (i.e. single wire) at a rate of 1 Mbps. These servos have a maximum holding torque of 1.618 N m and top speed of 306 degrees/s. The AX12s provide position, velocity and loading feedback, however velocity feedback is not used. Servo velocities are, instead, estimated in real time from position feedback because velocity readings provided by the AX12 is relatively noisy by comparison.

Commands are sent to the servos via an aggregate command packet which contains goal-position values for all servo units. Feedback is collected from each servo using individual data-request packets. Servos respond to each request with a response packet containing a corresponding feedback value. Given the number of servos in the network; communication overhead; and the one-wire communication configuration, servo updates are limited to a maximum update rate of 50 Hz over a half-duplex communication line. Gathering feedback over the half-duplex communication bus is particularly expensive because feedback requests require that the host processor wait after each dispatched for a response from each targeted servo. Moreover, each request/response cycle must finish to completion before a feedback request is made to another servo on the communication bus.

A dedicated circuit has been designed for use with these servos which converts a full-duplex serial line to a half-duplex AX12 bus. The circuit uses a two-state tri-state buffer which is switched via a general-purpose I/O line. This switching circuit is integrated into the system's main power switching and distribution board. Each servo is powered via an fused, software-switched 12 V supply line.

XBEE Wireless Radio

An XBEE Wireless Radio, shown in Figure This could be the picture from before, is used for communication between the LB and an external computer ground-station. The radio is interfaced with the LB via 57600 baud serial connection. This radio has a range of outdoor range of 27 meters and a maximum one-way transfer-rate of 115200 bps. Transfer rates between the LB and ground station are currently being limited to 57600 bps to compensate for a lack of hardware flow-control, which is required for stable, two-way communication between two XBEE radios at maximum communication rates. However, the selected communication rate is more than adequate for transferring necessary control information to and from the system without the need for additional flow-control hardware.

This wireless endpoint is used currently used interchangeably with the ODROID-XU's wireless WiFi radio, but will soon be retired to simplify hardware design and increase the platform's data streaming capabilities by switching to a WiFi-based line of communication. Ground station software, as well as the system's internal command-routing and networking software, is designed in such a way to easily accommodate this change.

2.3.2 Device Networking

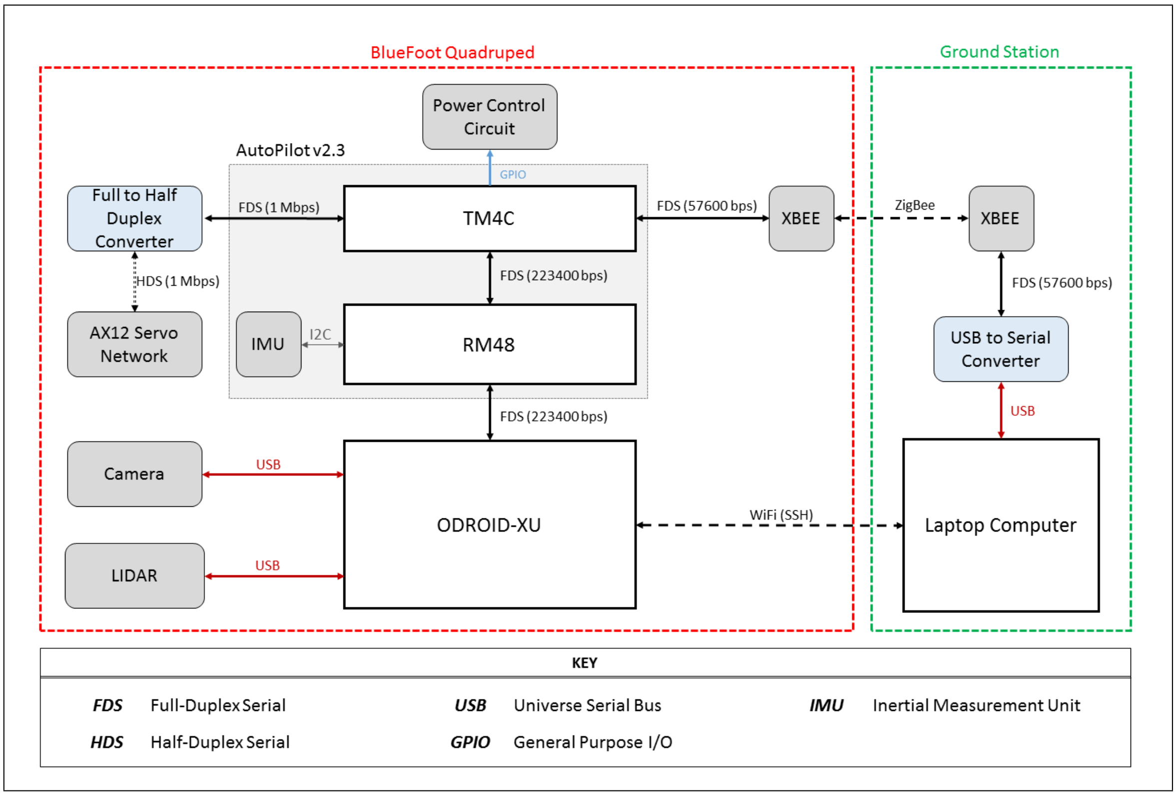


Figure 9: BlueFoot device networking diagram.

Figure 10 depicts how each major device is connected within the system and details the communication rates (f_{com}) between networked devices. Accompanying specifications are detailed in Table 2, which summarizes all device communication pairs and their corresponding baud rates.

$\mathbf{Port}_A$	$\mathbf{Source}_A$	$\mathbf{Port}_B$	$\mathbf{Source}_B$	f_{com}, **kbps**
UART0	TM4C	DIN/DOUT	XBEE*	55.7
UART2	TM4C	DIN+	AX12 Net.	1000
UART1	TM4C	LINSCI	RM48	223.4
SCI	RM48	USB (FTDI)	ODROID-XU	223.4
USB	ODROID-XU	DIN/DOUT	LIDAR	115.2
USB	ODROID-XU	USB	Cameras	No Spec.

Table 2: System communication port-pairs and corresponding data transfer rates XBEE* refers to the on-board XBEE module which communicates with the ground station.

2.4 System Power

2.4.1 Power Routing

System power routing is handled via an integrated power switching and distribution board. This board includes physical, main power switches which connects external power to two main, internal 12 volt buses for computer power (Net-1) and motor power (Net-2), respectively. The board also regulates system input voltage to a 5 V bus for use with on-board ICs and 3.3 V bus for powering the XBEE radio. Regulated power and power the servo motors of each leg controlled via three, two-channel power-switching IC's, which are toggled using six digital I/O pins on the TM4C processor of the LB. These power-switching chips allow for software-controlled power configuration, and further, software controlled emergency power cutoff to the servo motors. System main power is supplied via four 12 V (3 cell), 2 Ah Lithium Polymer battery packs.

2.4.2 Energy Requirements and Runtime

The power consumptions of BlueFoot's component device's are summarized in Table 3, which provides the operating voltage, V_{op}, and nominal current draw, I_{nom}, of each active, on-board component. Table 4 details battery specifications (output voltage and amp-hour rating) and BlueFoot's estimated run-time under nominal operating conditions.

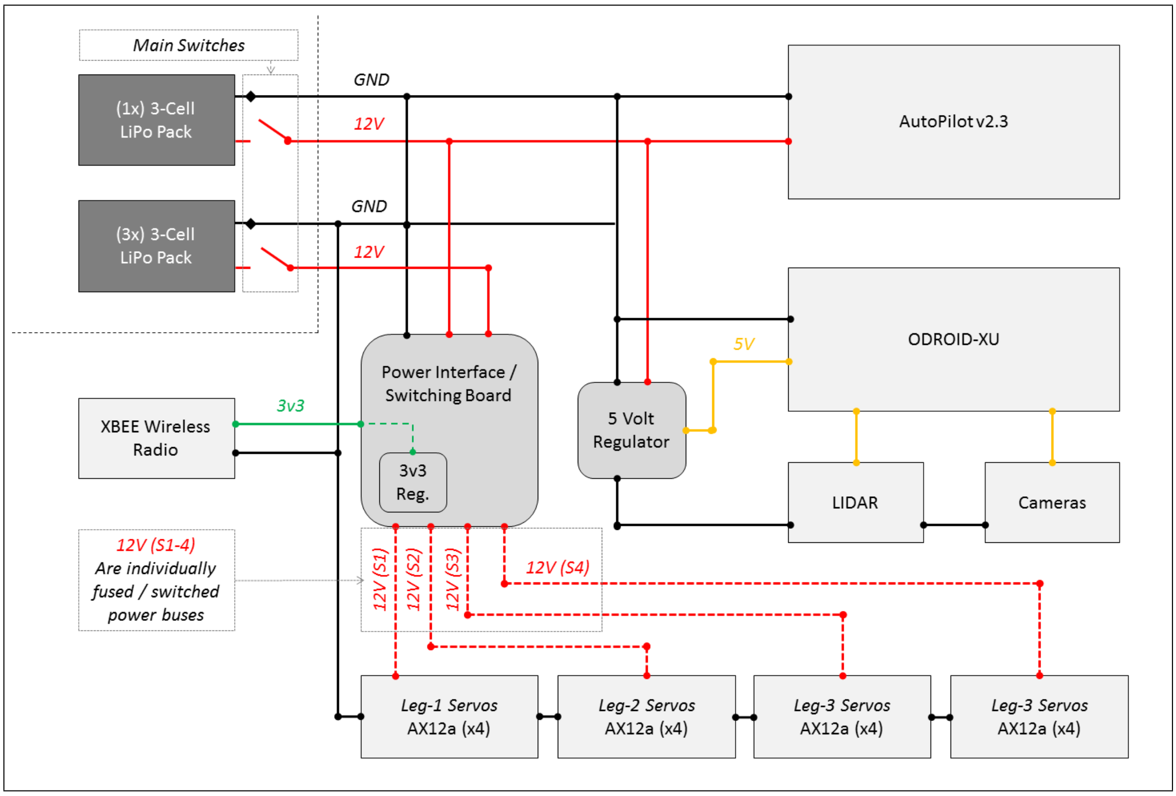


Figure 10: BlueFoot power routing.

Net	**Device**	V_{op}, V	I_{nom}, A
1	AutoPilot (RM48, TM4C)	12.0	0.40
1	XBEE Radio	3.3	0.25
1	ODROID-XU	5.0	2.0
1	Logitech-9000	5.0	0.1
1	Hokuyo-URG	5.0	0.5
2	AX12a Servos (x16)	12.0	9.6 (0.6 ea.)

Table 3: Power consumption summary by device (for single-camera configuration).

Net	**Battery Pack**	V_{out}, V	**Rating,** $A.hr$
1	3S LiPo Pack (x1)	12.0	2.0
2	3S LiPo Pack (x3)	12.0	6.0
Total Estimated Runtime		35-40 minutes	

Table 4: Battery power supply and estimated runtime summary.

CHAPTER III

Software

3.1 System Software Architecture

BlueFoot is controlled using a multi-processor software architecture which incorporates several independent core programs. Each of these programs handles portions of system control in a cooperative fashion. Core programs are executed on physically seperate operating units, allowing for low-level tasks, such as actuator command and feedback handling, battery monitoring, etc., to be decoupled from more computationally heavy tasks, such as high-level planning and navigation. With task-decoupling in mind, BlueFoot's software architecture is designed such that core programs could be readily offloaded to physically separate computing modules. Each of these control modules handles their own set of assigned tasks in independent control loops. Information is forwarded from each independent processor to update the overall BlueFoot software macro-system in an asynchronous fashion. System control tasks essential to BlueFoot's overall operation are divided into four main categories, which can be summarized as follows:

- *Low-Level Control* : power monitoring / switching, actuator command handling, communications routing, sensor data acquisition, script parsing and evaluation
- *Locomotion Control* : gait planning, gait adaptation, trunk pose adaptation
- *High-Level Control* : perception, motion planning, surface reconstruction, navigation, localization
- *Human-Operator Control* : joystick/keyboard commands, scripting commands

Low-level and locomotion control tasks are handled, exclusively, by the *Lower Brain* (LB), which designates a collection of software spanning over the RM48 and TM4C processors of the AutoPilot. High-level control tasks are handled by the Upper Brain (UB), which is a collection of software which runs on the ODROID-XU module. Lastly, a human operator can interface with the system wirelessly from a personal computer running ground-station software. The ground station communicates with the system through wireless communication lines routed to the TM4C processor and the ODROID-XU computer. Interfacing between the ground-station and the ODROID-XU is performed over an SSH connection. This secondary wireless connection is mainly used for configuring runtime settings and collecting data stored on-board the robot.

Since this software architecture is distributed over several separate computational units, an integral part of this control architecture is an efficient, reconfigurable inter-processor communication protocol. Namely, BlueFoot utilizes data packets transfered over serial lines to update system states between processors. These data packets are formatted using an in-house designed binary-XML protocol, called EXI. This protocol facilitates a highly customizable packeting structure for asynchronous inter-module communication and utilizes robust packet-error checking routines. This sections will detail BlueFoot's interprocessor communication protocol, namely the composition of packets transferred between processor.

This section will also detail the specific software-level tasks handled by each of BlueFoot's processor; the speed at which each core software element is run (update frequency); and what data must be communicated between software elements for operation. Additionally, this section will describe the ground-station software and corresponding user-interface used to control the BlueFoot Quadruped and administer high-level commands.

3.1.1 System Task Allocation

Figure 11 depicts how core software and associated control elements are related within the BlueFoot software macro-system. This section will detail the tasks carried out by each major software module implemented on the BlueFoot quadruped.

TM4C (Lower Brain)

As previously mentioned, the TM4C processor on-board the AutoPilot module is responsible for *Low-Level* tasks and can be viewed as a safety / communications routing co-processor. Within its main program loop, the TM4C polls the system's main battery voltage via an ADC interface; handles communication (packet decoding) routines between the ground-station and the RM48 system nodes; and handles command dispatching and feedback polling with the system's 16 servo actuators. The TM4C is

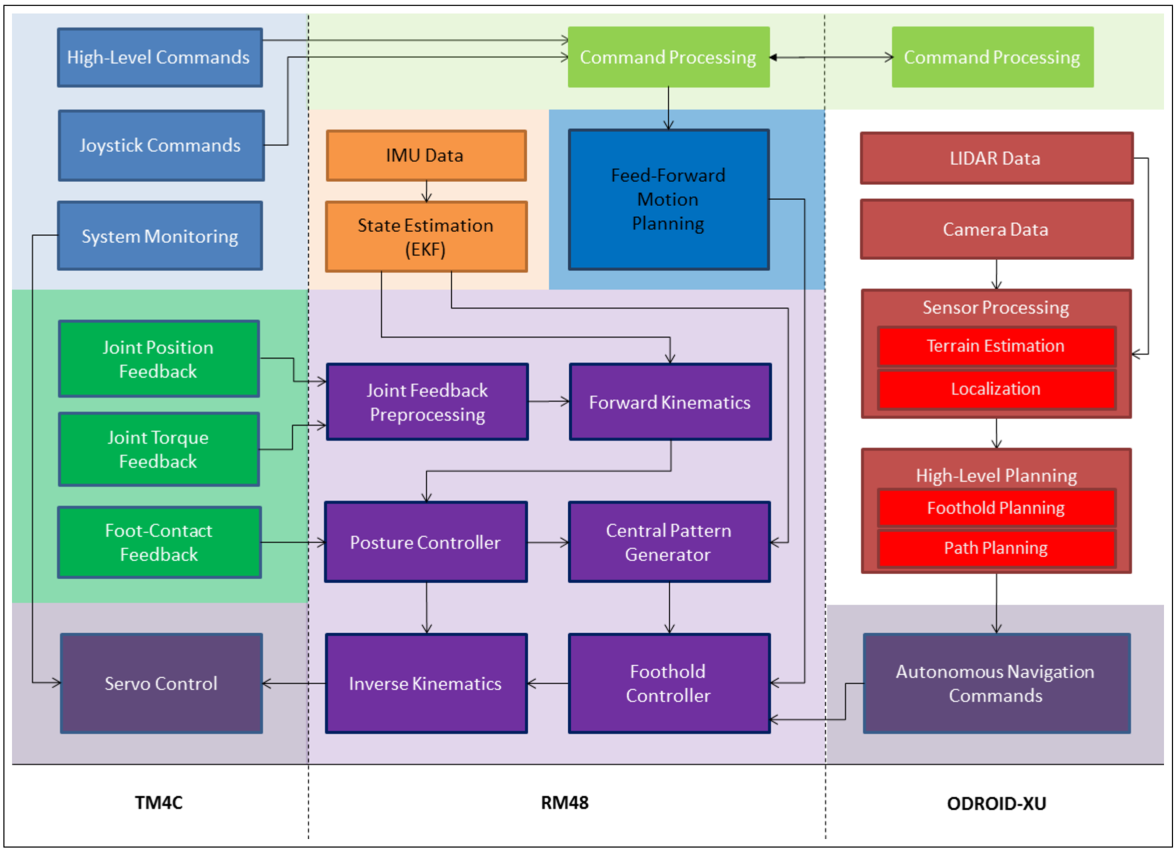


Figure 11: BlueFoot's on-board processes/signals and their relationships.

directly interfaced with two dual-channel power switching IC's and is used to control the supply of power to each leg. Power is controlled by toggling general purpose IO pins in software. The state of these pins is administered as part of a periodic packet command/update packet sent from the ground-station. Since the TM4C has this control over the system's actuators (which consume most of the system's power) and battery monitoring capabilities, it runs a safety routine which is responsible for halting motor activity and cutting system power on low-battery or power-fault conditions, as well as during unexpected breaks in communication with the ground-station.

As previously mentioned, the TM4C handles communication routing between system processing modules; as well as with the controllers on-board each smart-servo. Administering servo commands and collecting servo feedback it the TM4C's highest priority task. This process, which involves both commanding and requesting feedback from each servo, is relatively expensive and limits the TM4C's main-loop frequency to roughly 50 Hz. Thus, it is particularly important that this task is offloaded to this processor. Other safety and communication-related tasks are much less expensive, by comparison, and allow the servo actuators to be updated quickly as possible without encumbering other system control operations.

RM48 (Lower Brain)

The RM48 is responsible for several *Low-Level* tasks, including IMU polling and handling communication with the TM4C and ODROID-XU. Each set of collected IMU data is passed along to an Extended Kalman Filter (EKF) routine, which is used to estimate that orientation of the BlueFoot's trunk.

The RM48's primary function is to carry out motion control and gait-planning tasks. To achieve this, the RM48 handles a state machine which switches between planned motion execution and trajectory control; and a Central Pattern Generator (CPG) based gaiting controller, which will be discussed in more detail in Chapter V. Additional functions for body and posture (position and orientation) control, including trunk control procedures and gait-stabilization routines are run in tandem with the aforementioned gait-control task.

Motion and gait controls, which are performed in the robot task-space, are converted into joint-space reference angles, q^r, via an inverse kinematics (IK) routine. The IK routine is executed at all times when the legs are engaged for the purpose of issuing servo position commands. The RM48 also maintains BlueFoot's forward kinematic model, which is used to estimate the position of each foot relative to the trunk. This model relies on the EKF-generated trunk orientation estimate, $\hat{\theta}_b$, and joint position feedback, q. BlueFoot's inverse and forward kinematics models will be detailed in Chapter IV. The RM48 runs its full control loop at approximatively 100 Hz (twice the speed of the TM4C control loop) to facilitate higher integration stability when updating gait related controller dynamics, dynamic motion controls, task-space reference trajectories.

Lastly, the RM48 handles an on-board scripting engine (based on the MIT Squirrel scripting language), which interprets lexical commands. This scripting engine is capable of handling a large number of high-level commands and is complex enough to handle function and class definitions in real time [47]. The scripting engine currently being used to evaluate BlueFoot's core user-command set, ranging from simple state toggling and parameter modification, to the prescription of user-specified way-points for navigation, among other high-level command items. Scripting commands are passed from the ground station (via a text-input terminal) and routed through the TM4C to the RM48, where they are finally evaluated.

ODROID-XU (Upper Brain)

The ODROID-XU runs software upon the Debian (Linux) operating system distribution "Jessie." The use of an Linux operating system extends itself to a number of conveniences, such as to ability to run several tasks in parallel threads. Inbuilt USB-serial drivers are used to acquire data from USB-interfaced vision sensors. Namely, the ORDOID runs sensor handling elements used for acquiring and buffering camera images and controlling camera frame-rate; as well as acquiring LIDAR scans. The ORIOID uses these sensor inputs to perform several navigation-related tasks. These tasks will be described in more detail in Chapter VI.

The ODROID-XU utilizes 2D-LIDAR scans (frames) and trunk-pose estimates to form organized 3D point clouds. These point-clouds are further processed to reconstruct 3D terrain surfaces and height-maps, which are then used for step-planning. LIDAR and camera data is utilized in a potential fields-based navigation routine. Image processing and image feature detection runs as a separate process on the ODROID. In particular, the open-source libraries OpenCV (Open Computer Vision Library), OpenPCL (Open Point-Cloud Library), and Boost are heavily used in the software developed to carry out the aforementioned tasks [45, 48, 49]. Collectively, software written for this platform was generated using a mixture of C++ and Python.

As previously mentioned, the ODROID-XU can handle a limited set user-commands on its own, which are administered directly to the ODROID-XU from an SSH terminal on the ground-station computer. These commands include core-program start-ups and runtime configurators. Essentially, the ODROID's software core is designed as a completely independent software module which replaces the roll of a human director, as it handles the bulk of the systems high-level planning and navigation tasks. Moreover, if the ODROID-XU is removed from the BlueFoot system, the system can still be operated via remote-control heading commands provided from human operator using BlueFoots ground-station joystick control inputs.

3.1.2 Inter-processor Communication

This section will detail the contents of the data packets transfered between processors, which is summarized in Figure 12. System directives, generated by a human operator who interacts with the robot via a graphical user interface and joystick controls, originate from a ground-station computer. Update packets which are sent from the ground-station to the BlueFoot robot (TM4C) are composed as shown in Table 5:

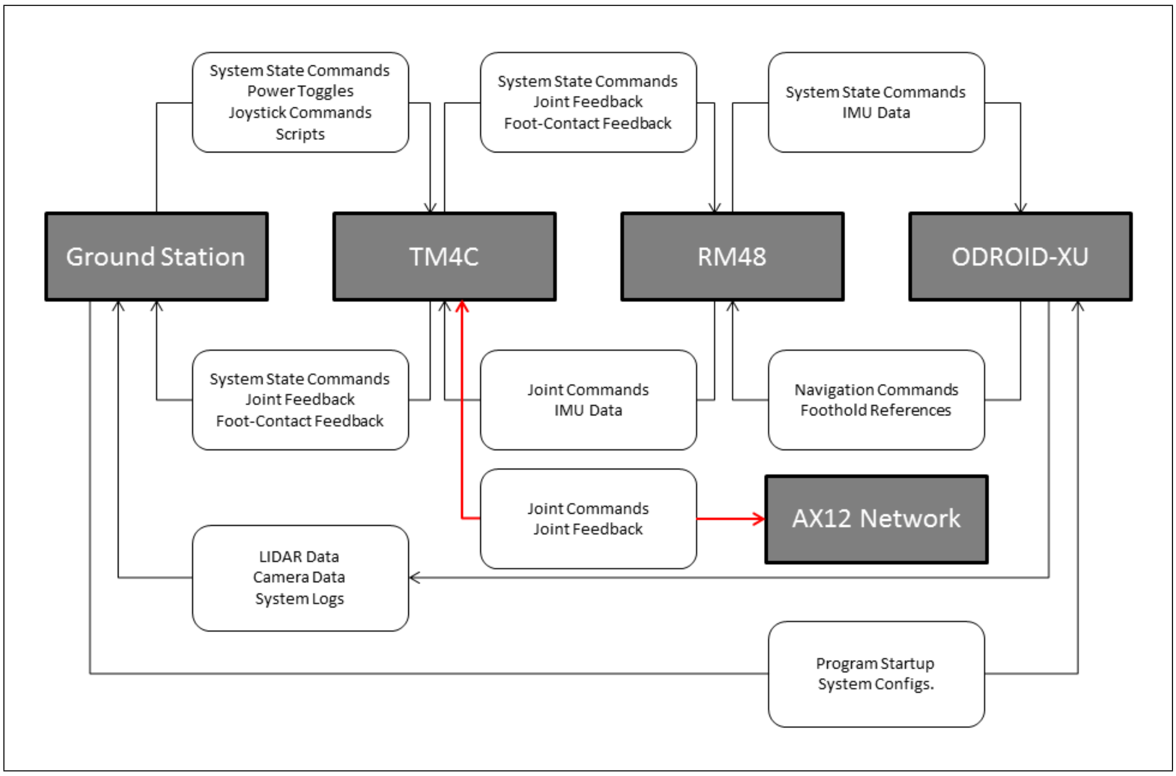


Figure 12: Communication flow between processors on the BlueFoot Platform.

32-bits	*8-bits*	*8-bits*	*16-bits*	*16-bits*
HEADER	Master-Tog.	Power-Tog.	Unused	Network Info
Variable				
Scripts				

Table 5: Structure of the packets sent from Ground-Station to TM4C.

Every packet issued has a 4-byte (32-bit) header, which is added as part of the EXI protocol. As part of the internal system protocol, every packet contains a "Master Status Vector", which is comprised of a fixed length, 7-byte sequence of essential system information and control states. The "Master Toggles" section (first 8-bits after header) enumerates major systems states, including *On-line, Standby, Off-line* and *Suspended* system state designations. The next 8-bits are used to toggle on-board power, namely the power supplied to each leg. The remaining 32-bits are used for specifying battery voltage (8-bits), power-fault states (8-bits), generic binary feedback toggles (8-bits), and system networking information (16-bits). The last section of this packet contains scripting commands, which can be of varying lengths. For example, forward velocity and turning rate commands (gathered form a joy-stick controller) are administered in the form of scripted commands.

Packets sent from the TM4C to the Ground-station contain status status item which are maintained on-board the robot, and appear as shown in Table 6: The *Joint Pos. FB* (joint position feedback) element is composed of 32-bytes corresponding to the joint position feedback from each actuator. To avoid redundancy, the structure of the packets communicated from the TM4C to the RM48 and vice-versa will not be depicted explicitly. These packets contain a 7-byte master status vector with only the *Master-Toggle* and *Network Info* fields populated. The TM4C sends the same joint feedback information to the RM48 as it does the Ground Station. For packets sent from the TM4C to the RM48, this field is replaced by a 32-byte sequence of joint-position commands.

32-bits	*8-bits*	*8-bits*	*8-bits*	*16-bits*
HEADER	Master-Tog.	Unused	Foot-Contacts	Network Info
2048-bits				
Joint Pos. FB				

Table 6: Structure of the packets sent from the TM4C to the Ground-Station.

Packets sent from the RM48 to the ODROID-XU contain additional fields which contain dynamical-state information used in planning on the ODROID. State information is sent in the form of vectors with single precision floating-point (32-bit) elements. Such information includes trunk orientation estimates, angular rate, and global position (generated from open-loop command integration); and four foot-position estimates, each of which are represented as a separate 3-element vector. These packets have a structure which is depicted in Table 7. Packets sent form the ODROID-XU to the RM48 contain command items, such as forward velocity, turning rate and trunk-pose commands, as

32-bits	*8-bits*	*8-bits*	*8-bits*	*16-bits*
HEADER	Master-Tog.	Unused	Foot-Contacts	Network Info
96-bits	*96-bits*	*328-bits*	*328-bits*	
Orientation	Angular Rate	Trunk Pos.	Foot Positions	

Table 7: Structure of the packets sent from the RM48 to the ODROID.

well as foothold-references and corrected global trunk position estimates. These packets are constructed as shown in Table 8 :

32-bits	*8-bits*	*8-bits*	*8-bits*	*16-bits*
HEADER	Master-Tog.	Unused	Unused	Network Info
32-bits	*32-bits*	*96-bits*	*328-bits*	*96-bits*
Velocity	Turning Rate	Trunk Pose	Footholds.	Trunk Pos.

Table 8: Structure of the packets sent from the ODROID-XU to the RM48.

3.2 Ground Station

Ground-station software used for controlling the BlueFoot platform is written in C++ using an open-source graphical user interface design library called *wxWidgets* [50]. *wxWidgets* is used, primarily, for the ground-station's front-end design. Namely, the ground-station code is designed such that that its UI design is reconfigurable. This is made possible by an XML-based design configurator, which can be used to change the look and location of buttons and panels which make up the UI, without having to modify the ground-station back-end software.

The back-end portion of the ground-station software handles interrupts generated by user inputs, such as button presses and text input. *wxWidgets* make the process of associating functional callbacks to use input events relatively easy. Additionally, *wxWidgets* provides an interface for collecting joystick inputs, which are interpreted and sent to the BlueFoot as remote-control commands. *Boost*, a C++ utility library, is utilized to handle socket and serial-port IO controls. This library is particularly important handling serial-IO ports on the ground station computer, which are used in communicating with the robot over a wireless radio connection.

The ground station is composed of several main sections of UI which provide interfaces for administering system master-state, operating-state and power toggling; providing system commands; and scripting. Ground station commands are sent the the robot in order to modify operating states and administer manual navigation commands. Updates from the ground-station are sent to the platform rate of 25 Hz using the an XBEE radio module. BlueFoot replies to each ground-station update with a packet of internal configurations, as shown in Table 6, which is then used to check that system updates and commands have been received and that the system is live.

Namely, BlueFoot sends the following particles of system information back to he ground-station:

- battery voltage levels
- power fault conditions
- foot contact feedback
- and joint position feedback.

This data is used to update several key portions of the UI. Battery levels are used to update a battery meter, which indicates the current system voltage level. Additionally, a dynamic text box displays the system's power-fault state to the user. Joint position and foot-contact information is used to update a visualization of the robot in real time. Joint positions, foot-contact states and battery levels are time-stamped and automatically logged during each ground-station session.

CHAPTER IV

System Modeling

4.1 Kinematic Model

The kinematic model of the BlueFoot platform is paramount for trajectory planning, localization, and adaptation in the robot task-space. In particular, inverse position and velocity solutions are used to prescribe joint-space commands from particular foot trajectories planned in the world coordinate frame. Additionally, BlueFoot's forward kinematic model is utilized to estimate the position of each foot using the position and orientation of the trunk; and joint position feedback. This section will describe BlueFoot's forward and inverse kinematic models, as well as how these models are used in motion planning and control tasks.

4.1.1 Forward Position Kinematics

To formulate the kinematics model, a set of coordinate systems have been defined and are described by Figure 13. Note that the frame O_0 represents the world coordinate frame; and O_b is the coordinate frame, centered at p_b attached to the platform and is always aligned with O_0. O_b represents a body frame rigidly attached to the center of the trunk. The orientation and position of the trunk are defined by vectors of $\theta_b \in \Re^3$ and $p_b \in \Re^3$, which relate the frame O_b to the world frame O_0. Coordinate frames $O_{i,0}$ are attached to the first joint of each i^{th} leg. The j^{th} joint position of each i^{th} leg is represented by the points $p_{i,j}$ in the frame O_0. These spatial locations are generated from a combination of the body orientation, θ_b, and joint positions for each i^{th} leg, $q_i = [q_{i,1}, q_{i,2}, q_{i,3}, q_{i,4}]^T$. $q_{i,1}$ represents the position of the hip-joint (joint closest to the center platform), which rotates in the direction of the transverse body plane. The joint variables $q_{i,2}, q_{i,3}$ and $q_{i,4}$ represent the lateral-hip, knee and ankle joint rotations, respectively.

The coordinate transformation between world coordinate frame, O_0, and the zeroth Denavit-Hartenberg (DH) coordinate frame of leg i, $O_{i,0}$ (located at the origin of joint-1),

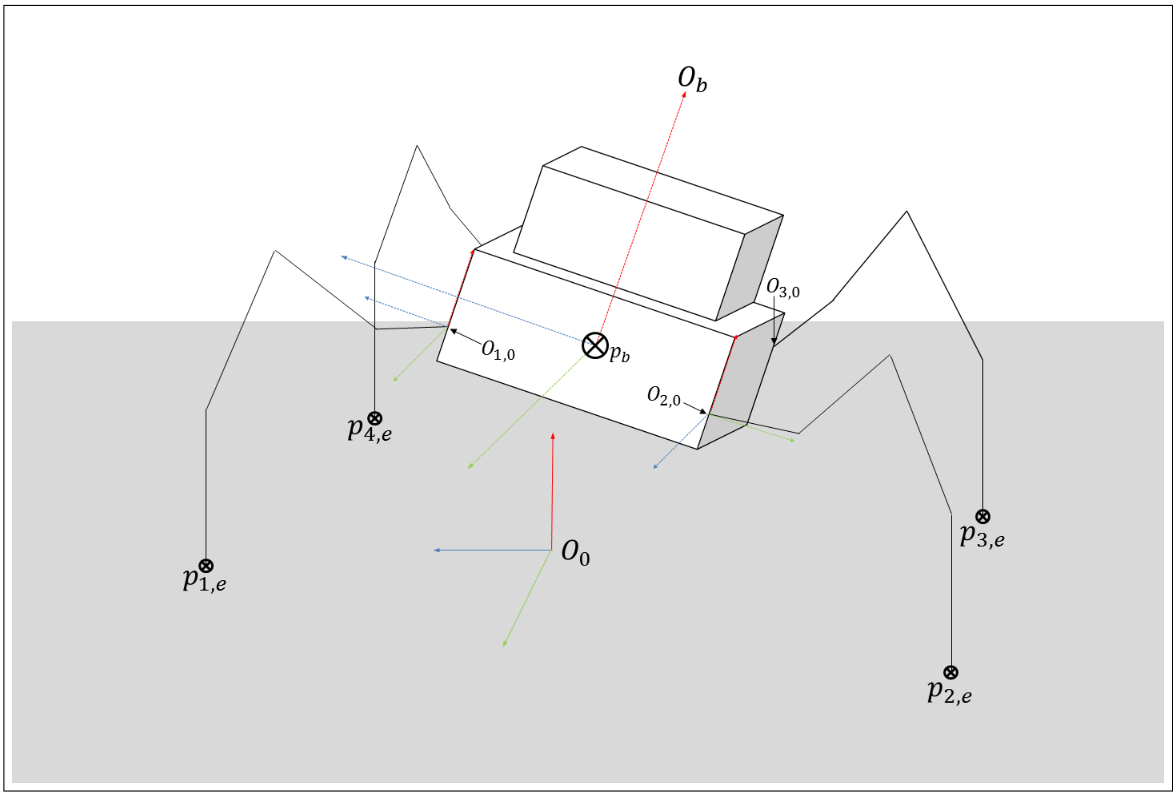


Figure 13: A visualization of BlueFoot's coordinate frame configuration.

is given by

$$H_0^{i,0} = \left[\begin{array}{c|c} R_{zyx}(\theta_b)R_z(\sigma_i) & R_{zyx}(\theta_b)\nu + p_b \\ \hline 0 & 1 \end{array}\right] \tag{4.1}$$

where $\sigma_i \equiv \frac{\pi}{2}(i-1) + \frac{\pi}{4}$ and $\nu \equiv R_z(\sigma_i)\beta$ with β defining an offset from O_b to where the first joint of each leg is attached to the body. R_{zyx} represents a rotation associated with the pitch (x-axis), roll (y-axis), and yaw (z-axis) angles of the main body about the platform frame, θ_b. R_z represents a rotation about the (z-axis) of the frame O_b.

A transformation from the zeroth DH frame of the to the $(j+1)^{th}$ joint of leg i is described, in general, by

$$H_{i,j}^{i,0} = \left[\begin{array}{c|c} R_{i,j}^{i,0} & p_{i,j}^{i,0} \\ \hline 0 & 1 \end{array}\right]. \tag{4.2}$$

The kinematics of each leg are identical. Thus, the transformations $H_{i,j}^{i,0}$ are of the same form and are derived by the DH parameters given by Table 9. Actual values for the link lengths a_{1-4} and body-offset, ν, are provided in Table 1.

Using these DH parameters, the transformations $H_{i,1}^{i,0}$, $H_{i,2}^{i,0}$, $H_{i,3}^{i,0}$, and $H_{i,4}^{i,0}$ can be

Link	a_i	α_i	d_i	θ_i
1	a_1	$\pi/2$	0	$q^*_{i,1}$
2	a_2	0	0	$q^*_{i,2}$
3	a_3	0	0	$q^*_{i,3}$
4	a_4	0	0	$q^*_{i,4}$

Table 9: DH parameters for all legs.

computed explicitly as follows:

$$H^{i,0}_{i,1} = \left[\begin{array}{ccc|c} c_{1,i} & 0 & s_{1,i} & c_{1,i}a_{1,i} \\ s_{1,i} & 0 & -c_{1,i} & s_{1,i}a_{1,i} \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.3}$$

$$H^{i,0}_{i,2} = \left[\begin{array}{ccc|c} c_{1,i}c_{2,i} & -c_{1,i}s_{2,i} & s_{1,i} & c_{1,i}(a_{1,i} + a_2 c_{2,i}) \\ s_{1,i}c_{2,i} & -s_{1,i}s_{2,i} & -c_{1,i} & s_{1,i}(a_{1,i} + a_2 c_{2,i}) \\ s_{2,i} & c_{2,i} & 0 & a_2 s_{2,i} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.4}$$

$$H^{i,0}_{i,3} = \left[\begin{array}{ccc|c} c_{1,i}c_{23,i} & -c_{1,i}s_{23,i} & s_{1,i} & c_{1,i}(a_{1,i} + a_2 c_{2,i} + a_3 c_{23,i}) \\ s_{1,i}c_{23,i} & -s_{1,i}s_{23,i} & -c_{1,i} & s_{1,i}(a_{1,i} + a_2 c_{2,i} + a_3 c_{23,i}) \\ s_{23,i} & c_{23,i} & 0 & a_2 s_{2,i} + a_{3,i} s_{23,i} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.5}$$

$$H^{i,0}_{i,4} = \left[\begin{array}{ccc|c} c_{1,i}c_{234,i} & -c_{1,i}s_{234,i} & s_{1,i} & c_{1,i}(a_{1,i} + a_2 c_{2,i} + a_3 c_{23,i} + a_4 c_{234,i}) \\ s_{1,i}c_{234,i} & -s_{1,i}s_{234,i} & -c_{1,i} & s_{1,i}(a_{1,i} + a_2 c_{2,i} + a_3 c_{23,i} + a_4 c_{234,i}) \\ s_{234,i} & c_{234,i} & 0 & a_2 s_{2,i} + a_3 s_{23,i} + a_4 s_{234,i} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.6}$$

The position of joint 1 of leg i in O_0, $p_{i,1}$ may now be computed with respect to frame O_0 by:

$$p_{i,1} \equiv E_p H^{i,0}_0 e_p. \tag{4.7}$$

where

$$E_p = [I_{3\times 3}, 0_{3\times 1}]$$
$$e_p = [0_{1\times 3}, 1]^T.$$

The position of joints 2-4 of leg i, may now be computed with respect to frame O_0 by:

$$p_{i,j} \equiv E_p H_0^{i,0} H_{i,(j-1)}^{i,0} e_p. \quad \forall \quad j \in \{2, 3, 4\} \tag{4.8}$$

Finally, the position of the end-effector (foot) of i^{th} leg. $p_{i,e}$, is achieved as follows:

$$p_{i,e} \equiv E_p H_0^{i,0} H_{i,4}^{i,0} e_p. \tag{4.9}$$

As a previously mentioned, BlueFoot's forward kinematic solution is used most prevalently in the estimating the position of each foot. Given estimates for body position and orientation, $\hat{p}_b$ and $\hat{\theta}_b$, a foot position estimate is explicitly define as a random variable:

$$\hat{p}_{i,e} = p_{i,e} + \Delta p_{i,e} \tag{4.10}$$

where $\Delta p_{i,e}$ is a random error which arises from variations due to noise in sensed joint positions, as well as error in the estimates $\hat{p}_b$ and $\hat{\theta}_b$.

A Note about Foot Localization in the Robot-Relative Frames

Using the previously defined forward kinematics model and the relationships defined in (4.8) and (4.9), the position of each joint and foot can be computed assuming the p_b and θ_b are known (or can be estimated) in the frame O_0. Provided inertial feedback, trunk orientation, θ_b, can be estimated by the use of an Extended Kalman Filter (EKF). p_b, however, requires more sophisticated localization measures, such as a Simultaneous Localization and Mapping (SLAM) scheme or absolute positioning via overhead camera or GPS. In lieu of an implementation for such a localization routine, it is convenient to generate foot positions estimate in a *robot-relative* frame, $O_{b'}$. This frame is not rigidly attached to the robot but moves along with the robot in O_0 according to the commanded translational velocity, v^r, and turning rate. ω^r, administered to navigate the system. These values will be described in more detail in Chapter VI. Moreover, this frame has its own position relative to O_0, defined by the translation $p_0^{b'}$. $O_{b'}$ is constantly aligned to O_0, with respect to rotation, making the rotation matrix which relates $O_{b'}$ to O_0 the identity matrix.

In this frame, the trunk position is regarded as an offset from the origin of $O_{b'}$, $p_b^{b'}$. Since there is zero rotation between O_0 and $O_{b'}$, the trunk rotation vector $\theta_b^{b'}$ is directly equivalent to θ_b in O_0. Using these translation and rotation definitions in place of their world frame counterparts, the robot-relative joint and foot-positions, $p_{i,j}^{b'}$ and $p_{i,e}^{b'}$ of each

i^{th} leg can be computed by defining a transformation from the robot relative frame to the first DH-frame for each leg, as follows:

$$H_{b'}^{i,0} = \left[\begin{array}{c|c} R_{zyx}(\theta_b)R_z(\sigma_i) & R_{zyx}(\theta_b)\nu + p_b^{b'} \\ \hline 0 & 1 \end{array}\right] \tag{4.11}$$

which follows the same definitions in (4.1). The transformation defined in (4.11) is used as a direct replacement for the matrix $H_0^{i,0}$ in (4.8) and (4.9) in computing the corresponding positions, $p_{i,j}^{b'}$ and $p_{i,e}^{b'}$. Knowledge of these positions is useful for planning foot and body-placement which depend directly on the location relative foot and body positions, but do not necessarily depend on the position of the robot in the world coordinate frame.

4.1.2 Inverse Position Kinematics

A foot configuration is specified by its coordinates p_e and an ankle orientation, γ_i, which represents a rotation about the axis of rotation of the second joint (lateral hip). Given a desired platform configuration, $\{p_b, \theta_b\}$, and desired i^{th} foot configuration, $\{p_{i,e}, \gamma_i\}$, the inverse kinematics solution for each i^{th} leg, q_i, is derived to be:

$$\begin{aligned} q_{i,1} &= \cos(i\pi)\left(\frac{\pi}{4} - \psi_i\right) \\ q_{i,2} &= \tan^{-1}\left(\frac{\zeta_{i,z}}{\sqrt{\zeta_{i,x}^2 + \zeta_{i,y}^2}}\right) \mp \cos^{-1}\left(\frac{a_3^2 - a_2^2 - \|\zeta_i\|^2}{2a_2\|\zeta_i\|}\right) \pm \pi \\ q_{i,3} &= \mp\cos^{-1}\frac{\|\zeta\|^2 - a_2^2 - a_3^2}{2a_2a_3} \\ q_{i,4} &= \gamma_i - q_{i,2} - q_{i,3} \end{aligned} \tag{4.12}$$

where

$$\begin{aligned} p_{i,e}^{i,0} &= E_p\left(H_{i,k}^0\right)^{-1}\begin{bmatrix} p_{i,e} \\ 1 \end{bmatrix} \\ \psi_i &\equiv \tan^{-1}\left(\frac{\left[p_{i,e}^{i,0}\right]_y}{\left[p_{i,e}^{i,0}\right]_x}\right) \\ \zeta_{i,x} &\equiv \left[p_{i,e}^{i,0}\right]_y \sin(\psi_i) + \left[p_{i,e}^{i,0}\right]_x \cos(\psi_i) - a_4\cos(\gamma_i) - a_1 \\ \zeta_{i,y} &\equiv \left[p_{i,e}^{i,0}\right]_y \cos(\psi_i) - \left[p_{i,e}^{i,0}\right]_x \sin(\psi_i) \\ \zeta_{i,z} &\equiv \left[p_{i,e}^{i,0}\right]_z - a_4\sin(\gamma_i). \end{aligned} \tag{4.13}$$

Here, $p_{i,e}^{i,0}$ represents the position of each i^{th} foot with respect to the zeroth DH frame of each i^{th} leg; and $\left[p_{i,e}^{i,0}\right]_x$, $\left[p_{i,e}^{i,0}\right]_y$ and $\left[p_{i,e}^{i,0}\right]_z$ represent the x, y, and z axis coordinates of the vector $p_{i,e}^{i,0}$. The selection matrix E_p is defined as it is in (4.7)-(4.9).

It is important to note that the ankle specification, γ_i, adds extra constraints on the system kinematics and, thus, reduces the number of inverse kinematics solutions per foot position to two.

Range of Motion

The range of articulation of BlueFoot's main body has been characterized with respect to a set of imposed joint limits (see Table 10) and has been performed with all feet fixed in a default stance (see Table 12). These limits have been selected according to the range of feasible angular position outputs of inverse position kinematics solution.

Joint Var.	$\theta_{i,1}$, rad	$\theta_{i,2}$, rad	$\theta_{i,3}$, rad	$\theta_{i,4}$, rad
Max Range	45^o	90^o	90^o	90^o
Min Range	-45^o	-90^o	-90^o	-90^o

Table 10: Imposed joint limits.

It should be noted that the imposed joint limits are soft limits that have been chosen because of the typical range of motion executed by each joint during locomotion. Physical joint limits are tabulated in Table 11.

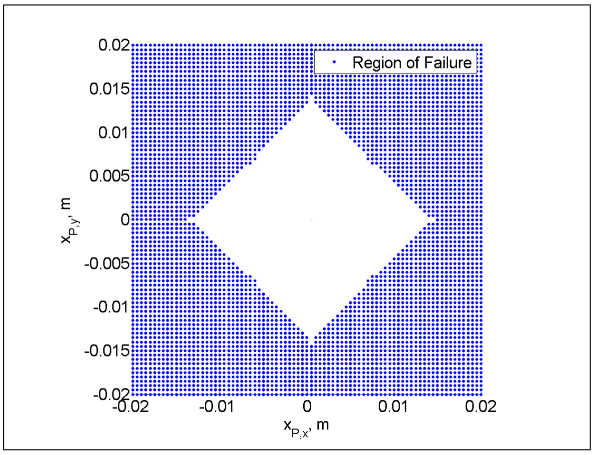
Joint Var.	$q_{i,1}$, rad	$q_{i,2}$, rad	$q_{i,3}$, rad	$q_{i,4}$, rad
Max Range	82^o	102^o	102^o	102^o
Min Range	-69^o	-102^o	-102^o	-102^o

Table 11: Physical joint limits.

Figures 14 depicts the maximum pitch and roll and region of planar motion that the main body can reach while in the default stance. The blue shaded region of each figure (labeled *Region of Failure*) depicts points at which joint positions exceed the imposed limits. It can be seen from Figures 14 that the platform can achieve a maximum pitch and roll of $\pm 37^o$and a total displacement of ± 1.4 cm along the x and y axis of $O_{b'}$.

	$p_b^{b'}$	$p_{1,e}^{b'}$	$p_{2,e}^{b'}$	$p_{3,e}^{b'}$	$p_{4,e}^{b'}$
x (m)	*	0.165	-0.165	-0.165	0.165
y (m)	*	0.165	0.165	-0.165	-0.165
z (m)	0.115	0	0	0	0

Table 12: Locations for the platform and feet when in the default stance, written with respect to the frame $O_{b'}$.



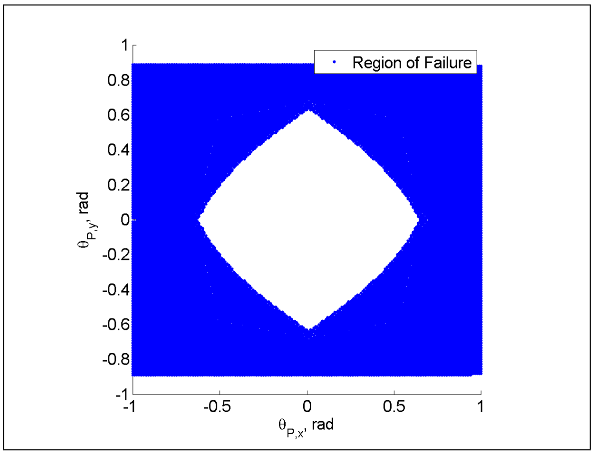


Figure 14: Range of motion for body's planar position (*left*) and body pitch and roll (*right*) while in the default stance.

4.1.3 Velocity Kinematics

Using the DH-coordinate transformation, $H_{i,4}^{i,0}$, the velocity kinematics of each i^{th} leg are derived as the Jacobian $J_{i,e}^{i,0} \in \Re^{6\times 4}$ where

$$\dot{x}_{i,e}^{i,0} = J_{i,e}^{i,0}\dot{q}_i \tag{4.14}$$

with $\dot{x}_{i,e}^{i,0} \in \Re^6$ being stacked vector of translational and rotational velocities, $\dot{p}_{i,e}^{i,0} \in \Re^3$ and $\dot{\theta}_{i,e}^{i,0} \in \Re^3$, respectively, of each i^{th} foot with respect to frame $O_{i,0}$. The matrix $J_{i,e}^{i,0}$ is defined explicitly in Appendix A.

Assuming the translational and rotational velocity of the trunk, $\dot{p}_b$ and $\dot{\theta}_b$, respectively; and the translational and rotational of each i^{th} foot, $\dot{p}_{i,e} \in \Re^3$ and $\dot{\theta}_{i,e} \in \Re^3$, respectively, are known, the translation velocity of each i^{th} foot can be written with respect to frame $O_{i,0}$ by:

$$\dot{p}_{i,e}^{i,0} \equiv \left(R_{i,0}^{0}\right)^T \left(\dot{p}_{i,e} - \dot{p}_b - S\left(\dot{\theta}_b\right)\left(p_{i,e} - p_b - R_{i,0}^{0}\vec{o}_\nu\right)\right) \tag{4.15}$$

where $\vec{o}_\nu = [\nu, 0, 0]^T$, $R^0_{i,0}$ is the rotation-matrix component of the transformation $H^0_{i,0}$ defined in (4.1), and $S(*)$ is a skew-symmetric matrix operator, which takes a 3×1 vector input. The corresponding rotational velocity of each i^{th} foot can be computed with respect to $O_{i,0}$ by:

$$\Theta^{i,0} \equiv \left(R^0_{i,0}\right)^T S\left(\dot{\theta}_{i,e} - \dot{\theta}_b\right) R^0_{i,0} = S\left(\dot{\theta}^{i,0}_{i,e}\right) \tag{4.16}$$

where the rotational velocity of each foot, described by $\dot{\theta}^{i,0}_{i,e}$, is recovered by:

$$\dot{\theta}^{i,0}_{i,e} \equiv \left[-\Theta^{i,0}_{1,2}, \Theta^{i,0}_{1,3}, -\Theta^{i,0}_{2,3}\right]^T \tag{4.17}$$

with $\Theta^{i,0}_{j,k}$ being the k^{th} element in the j^{th} row of the skew symmetric matrix $\Theta^{i,0}_{j,k}$ which results from (4.16).

Joint velocities required to attain $\dot{x}^{i,0}_{i,e}$ can be computed using $J^{i,0}_{i,e}$. However, since each of BlueFoot's of legs had 4 degrees of freedom, $J^{i,0}_{i,e}$ is rank deficient and $\dot{q}_i$ cannot be obtained by direct inversion. Instead, $\dot{q}_i$ can be approximated from $\dot{x}^{i,0}_{i,e}$ by a weighted, Penrose-Moore psuedo-inverse of $J^{i,0}_{i,e}$, which is defined as follows:

$$\begin{aligned} \dot{q}_i &\approx \left[J^{i,0}_{i,e}\right]^{\dagger}_{\Lambda_j} \dot{x}^{i,0}_{i,e} \\ \dot{q}_i &\approx \left(\left(J^{i,0}_{i,e}\right)^T J^{i,0}_{i,e} + \Lambda_J\right)^{-1} \left(J^{i,0}_{i,e}\right)^T \dot{x}^{i,0}_{i,e} \end{aligned} \tag{4.18}$$

where Λ_J is a strictly positive-definite weighting matrix that is typically chosen to have

$$\det(\Lambda_J) \ll 1.$$

The operator $[*]^{\dagger}_{\Lambda_j}$ defines a weighted pseudo-inversion off a matrix argument $(*)$ given a positive-definite weighting matrix Λ_j. A weighted pseudo-inversion is chosen over direct pseudo-inversion to avoid a non-smooth set of solutions $\dot{q}_i$. Degenerate solutions for $\dot{q}_i$ exist at particular manipulator configurations where $\left(\left(J^{i,0}_{i,e}\right)^T J^{i,0}_{i,e}\right)$ is non-invertible.

4.2 Dynamical Model

4.2.1 System State Vector and General-Form Dynamics

The dynamical model of the BlueFoot platform is treated a general, free-floating body with four ridged legs, each of which have four degrees of freedom. This system is fully described by the state vector $z \equiv [\eta^T, \dot{\eta}^T]^T \in R^{44}$ and its dynamics are:

$$M(\eta)\ddot{\eta} + C(\eta, \dot{\eta})\dot{\eta} + G(\eta) + \Delta H = \tau + J^T(\eta) f_{ext} \tag{4.19}$$

where $M(\eta)$, $C(\eta, \dot{\eta})$, $G(\eta)$ and $J(\eta)$ represent the system mass matrix, Coriolis matrix, gravity matrix and Jacobian, respectively [51]. ΔH has been included as a lump term to account for dynamical uncertainties, such as friction or unmodeled coupling effects. Additionally, $f_{ext} = [f_{1,ext}^T, f_{2,ext}^T, f_{3,ext}^T, f_{4,ext}^T]^T \in R^{24}$ represents a stacked vector of force-wrenches, $f_{i,ext} \in R^6$, applied to the system through each i^{th} foot. The state vector, η, can be partitioned as follows: $\eta = [p_b^T, \theta_b^T, q^T]^T$ with $p_b \in R^3$ and $\theta_b \in R^3$ representing the position and orientation, respectively, of the quadruped's trunk in an arbitrarily placed world coordinate frame, and $q \in R^{16}$ is a vector of joint variables, m of which are contributed by each leg. $\tau \in R^{22}$ represents a vector of generalized torque inputs and takes the form $\tau = [0_{1x6}, \tau_q^T]^T$ where τ_q represents a set of torque inputs to each joint. It is important to note that the states we are most interested in controlling, p_b and θ_b, are not directly actuated, and must be controlled via composite leg joint motions.

BlueFoot's dynamics, from (4.19), can be realized in a general, compact, state-space form by:

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= M^{-1}(z_1)(\tau + \Phi(z_1, z_2, f_{ext})) \\ \Phi(z_1, z_2, f_{ext}) &= J^T(z_1) f_{ext} - C(z_1, z_2) z_2 - G(z_1) - \Delta H \end{aligned} \tag{4.20}$$

where $z_1 = \eta$ and $z_2 = \dot{\eta}$. The notation $\Phi(z_1, z_2, f_{ext})$ is introduced for convenience as a composite dynamical term. This term will be referred to, simply, as Φ in the sections that follow. The system dynamics are also considered in an approximate, discrete-time (first-order) form as follows:

$$\begin{aligned} z_{1,k+1} &= z_{1,k} + (e_{1,k}^{\Delta_s} + z_{2,k})\Delta_s \\ z_{2,k+1} &= z_{2,k} + M_{1,k}^{-1}(e_{2,k}^{\Delta_s} + \tau_k + \Phi_k)\Delta_s \\ t &= \Delta_s k \end{aligned} \tag{4.21}$$

where $M_{1,k} = M(z_{1,k})$ and $\Delta_s \equiv (f_s)^{-1}$ with f_s defining a uniform sampling frequency in Hz. The terms $e_{1,k}^{\Delta_s}$ and $e_{2,k}^{\Delta_s}$ are used to explicitly account for system discretization errors, which vary with respect to the step-size, Δ_s.

4.2.2 Joint-Servo Dynamics

The motor dynamics driving each joint need to be considered for use in control design since the input to BlueFoot's servo motors at each joint is a reference position command. In model-based control schemes to follow, a simple model of the motor dynamics will be utilized. Moreover, servos are considered as simple torque generators of the following form:

$$\tau_q = k_s(q^r - q) \tag{4.22}$$

where $k_s > 0$ is a constant, scalar gain and q^r is a joint position reference. The servos being utilized to drive the leg joints of the BlueFoot quadruped have high-gain position feedback which allows us to model the motors, simply, as a static block which transform reference trajectories to torque outputs. All of these servos are identical, and thus have identical gains. One could instead consider the full motor dynamics for computing reference positions given a desired torque. The simple model stated above was adequate for achieving desired results with all proposed control schemes which use this torque-generator model.

4.3 BlueFoot Simulator

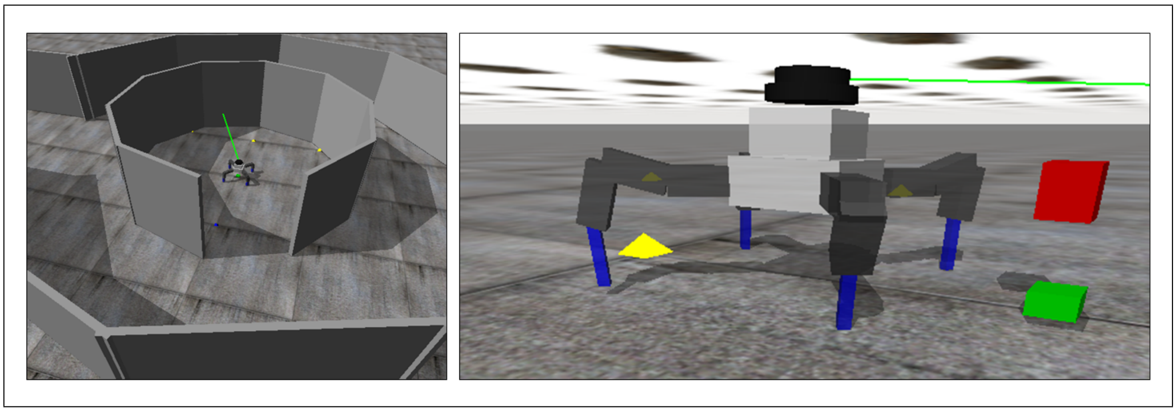


Figure 15: Visualization of the BlueFoot simulator.

The BlueFoot platform is comprised of 17 rigid bodies, 16 of which are linkages between joints; and a main platform. Since each limb is formed from four revolute joints, the system has a total of 22 DOF, 16 of which are actuated. The main platform's configuration is generated through particular configurations of the legs. Furthermore, because each linkage is constrained to a rotation about a single axis, the rigid-body dynamic model of the BlueFoot quadruped is represented by 44 states. These states represent the position and velocity of each joint, q_i and $\dot{q}_i$, respectively. Additionally, foot contact states are represented by binary scalars, $\mu_i \in (0, 1)$, which describes whether the foot is not in contact or in contact, respectively.

The dynamics of this system are somewhat complex because the quadruped's legs continuously make and break contact with the ground during gaiting. Additionally, the surface attributes and contact effects may vary significantly during gaiting and for various environments. A numerical dynamics engine, Open Dynamics Engine (ODE) [52], has been utilized to implement a dynamics simulator for the BlueFoot platform. The simulator allows for various reconfigurations of environmental parameters, such as joint-friction, body-contact friction, and physical obstacles. Additionally, this simulation environment is implemented with a variety of geometric collision solvers and and efficient collision search method.

The simulator's numerical engine, based on ODE is updated at 1000 Hz to attain high-fidelity, especially during contact phases, which require higher-than-normal integration stability to achieve reasonable simulation accuracy. The input to the simulator is the desired main body configuration (i.e., position and orientation); desired foot placements; and desired ankle orientations, from which an inverse kinematic model is solved to attain all the desired joint configurations for the legs. Servos at each joint have built-in, tunable proportional controllers that effectively render them as ideal torque generators responding to the command input, as depicted in (4.22). To mimic the behavior of the system, the commands to each joint motor are updated at 50 Hz, which matches the update rates at which the actual robot serves the P-controllers at each joint with reference positions. This rate has been chosen to account for the speed of inter-processor communications, and is adequate given the operating bandwidth of the robot.

The BlueFoot simulator utilizes a dynamical model of the platform which consists of purely rectangular bodies. To match the dynamics of the true system, the individual mass and inertia parameters of each rigid body have been generated using a 3D model analysis software. This software takes into account geometric irregularities and variation of materials when computing the inertia of each body. The simulator also contains IMU and LIDAR sensor models for gathering feedback from the simulated environment. Each sensor is modeled with appropriate measurement noise so as to more closely match the performances of the actual IMU and LIDAR sensors on-board the BlueFoot platform.

Careful, manual tuning was performed on simulated robot parameters, including joint update gains, joint-friction and surface-friction model coefficients to achieve closer matches between simulated results and the actual robot motions. A performance comparison can be seen in 16, which shows joint position outputs for all joints of the front-

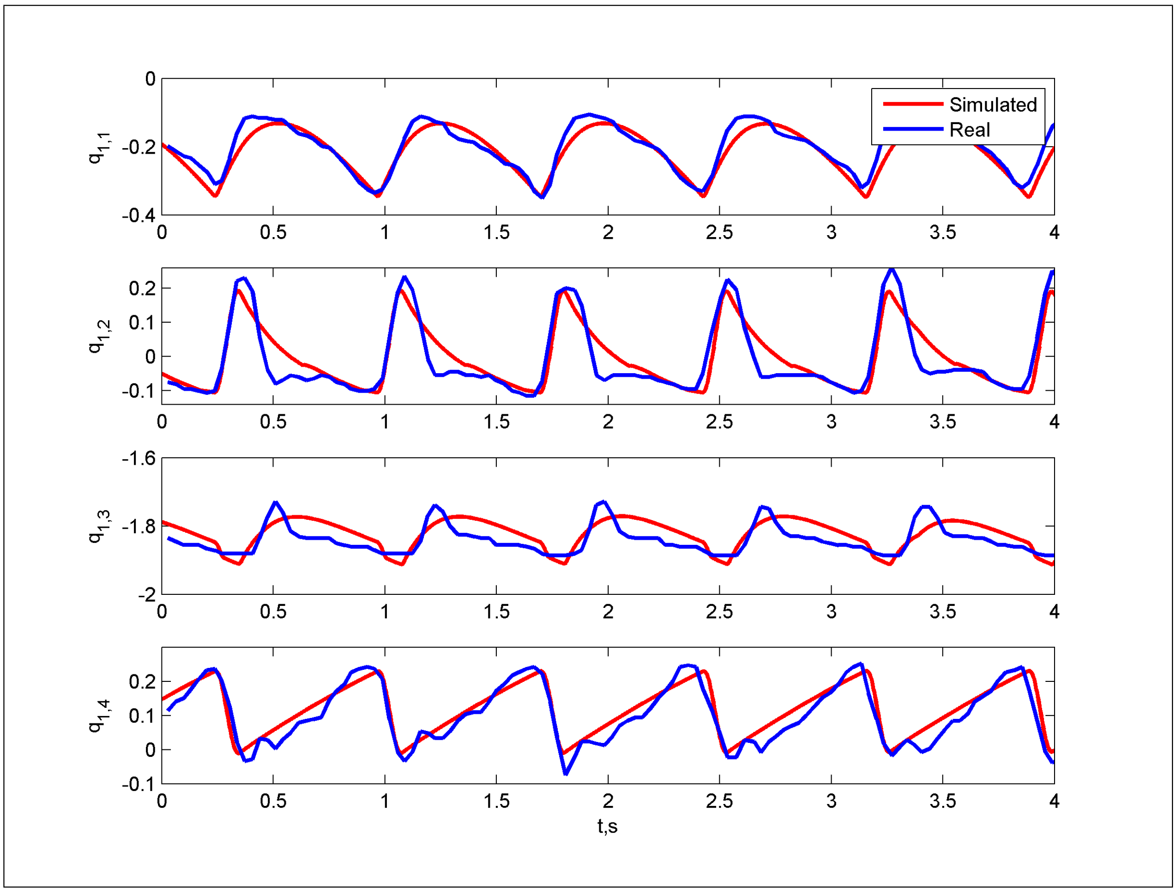


Figure 16: Joint position comparison between the simulated and real quadruped while performing a gait which achieves a forward velocity of $60\frac{mm}{s}$

right leg of the actual and simulated robot while performing a forward gait at 0.060 m/s. It can be seen in that the simulator provides and approximate representation true system performance within some margin of error. The discrepancies between the two data sets can be seen to usually occur at the minima of each series when the robot makes contact with the ground after a step. These differences can be attributed to ground-contact model inaccuracies. Work is still left to be done in improving the simulator's contact model, but this is more the fault of the simulation software libraries being used, namely in its low-order contact models. These contact models are very simplistic for the sake of simulation speed. Contact models could certainly be improved such that they more closely matches the effects of real-world surfaces. Additionally, discrepancies could also be attributed to the low-order joint model being used to represent the system's servo motors at each joint, described in (4.22). This model neglects the effects of higher-order frictional effects which modify the rotation profile at each joint. All of the other simulated leg joints exhibited similar results to the actual motions.

Despite these discrepancies, this simulation platform has proven to be adequate for use in testing gaiting and control algorithms which have been implemented on the real system. Moreover, all motion control algorithms which were first developed in this simulated environment have been ported to the real robot platform and performed in a highly comparable manner.

The simulator has also been utilized to model BlueFoot's on-board LIDAR unit for use in testing navigation and terrain mapping algorithms and associated motion routines. Laser hits are modeled as force-less collisions between a ray-type geometry, used to represent the laser beam of the LIDAR, and environmental elements. Because the ray geometry can penetrate environmental elements in multiple places, or even multiple elements within the environment at a single instance, some post processing is performed on collision point returned by the simulator such that the point of intersection *closest* to the robot is taken as a laser-beam hit. Additionally, sensor realism is imposed by limiting the data access frequency to buffered laser data by control routines operating the simulated robot to a rate of 10 Hz, which matches the scan rate of BlueFoot's on-board LIDAR. The angular velocity of the spinning laser head and the depth of each laser beam hit are corrupted by zero-mean Gaussian random noise with variances close to the parameters specified in the datasheet of the LIDAR unit.

CHAPTER V

Gaiting and Gait-Stability Control

5.1 Overview

Legged robotic systems have long employed motion controllers based on limit cycle oscillators and, more recently, Central Pattern Generators (CPGs) for the purpose of generating bio-mimetic gaits [11, 13, 15–17, 53–56]. Since these motion control methods are open-loop motion planners (*i.e.*, not inherently formulated to incorporate feedback) they do not perform any active system stabilization on their own accord. As a result, implementations involving these locomotion methods often require auxiliary control mechanisms which provide gait stability. Fixed point methods, which include considerations of the system's zero-moment point (ZMP) and center of gravity (COG), are utilized in the design of stable oscillator driven gaits. These methods are summarized in [5].

Developments in CPG-based gait controllers have led to the incorporation of "reflexive" feedback mechanisms aimed at correcting foot-placement during gaiting on uneven terrain or various surfaces. One such approach involves active compliance to each leg by directly modifying CPG oscillators units through feedback-driven modulations. In [6,17], a CPG for each leg is modified by a neural oscillator with one tuning parameter.

This chapter will detail a similar, reflex-adaptive CPG gait generation method which utilizes IMU feedback to modify CPG limit-cycle parameters. Additionally, this section will present related motion-control methods implemented on the BlueFoot platform which aid in gait stabilization, including a virtual force-based foot placement planner and a ZMP-based body placement controller; as well as a learning-based control technique used to level BlueFoot's trunk during gaiting.

5.2 Central Pattern Generator (CPG) Based Gaiting

A central pattern generator (CPG), which includes a reflexive mechanism using sensory feedback, is utilized as the core of the robot's gait generation algorithm. As previously introduced, CPGs are a form of neural oscillator network which mimic biological mechanisms for repetitive motor tasks [13, 54]. CPGs commonly consist of a network of multi-state unit-oscillators. These unit-oscillators are coupled such that the motion of one oscillator drives or attenuates the motion of other oscillators it is connected to, creating phase-locked limit cycles.

In this context, the limit-cycles generated with a CPG will be utilized to drive a quadruped walking gait by mapping oscillator outputs to foot position controls. CPGs are widely used in this way as they provide a compact method for prescribing rhythmic gaits with variable stepping sequences [15, 57, 58]. CPG-driven gait controllers are convenient as they allow for continuous transitioning between gaiting patterns through the modification of oscillator coupling. Reflexes are built into the oscillators which use IMU measurements to modulate the CPGs unit oscillators, as will be outlined later in this section.

The CPG implemented on BlueFoot consists of four modified two-state Hopf Oscillators connected through a coupling matrix K. The states of each i^{th} unit Hopf Oscillator are designated by the pair $\{y_{1,i}, y_{2,i}\}$. These oscillator states are stacked into the vectors $y_1 \in \Re^4$ and $y_2 \in \Re^4$, which are composed as follows:

$$\begin{aligned} y_1 &= [y_{1,1}, y_{1,2}, y_{1,3}, y_{1,4}]^T \\ y_2 &= [y_{2,1}, y_{2,2}, y_{2,3}, y_{2,4}]^T \end{aligned}$$

The oscillator output vector y_2, parameterizes the trajectories of each i^{th} foot. The resulting reflexive CPG system, written which respect to the stacked-state vectors y_1 and y_2, is described by

$$\begin{aligned} \dot{y}_1 &= A_1 \left(\Psi_M M\left(y_1, y_2\right) - \Gamma\right) y_1 + \Psi_\omega W y_2 \\ \dot{y}_2 &= A_2 \left(\Psi_M M\left(y_1, y_2\right) - \Gamma\right) y_2 - \Psi_\omega W y_1 + K y_2 \end{aligned} \tag{5.1}$$

where $M\left(y_1, y_2\right) \in \Re^{4\times4}$ is defined as

$$M\left(y_1, y_2\right) = \begin{bmatrix} y_{1,1}^2 + y_{2,1}^2 & 0 & 0 & 0 \\ 0 & y_{1,2}^2 + y_{2,2}^2 & 0 & 0 \\ 0 & 0 & y_{1,3}^2 + y_{2,3}^2 & 0 \\ 0 & 0 & 0 & y_{1,4}^2 + y_{2,4}^2 \end{bmatrix},$$

$A_1, A_2, \Psi_M, \Psi_\omega \in \Re^{4\times4}$ are constant oscillattor gain matrices; and Γ and W are diagonal matrices, defined by

$$\Gamma = \begin{bmatrix} \gamma_1 & 0 & 0 & 0 \\ 0 & \gamma_2 & 0 & 0 \\ 0 & 0 & \gamma_3 & 0 \\ 0 & 0 & 0 & \gamma_4 \end{bmatrix}, \quad W = \begin{bmatrix} \omega_1 & 0 & 0 & 0 \\ 0 & \omega_2 & 0 & 0 \\ 0 & 0 & \omega_3 & 0 \\ 0 & 0 & 0 & \omega_4 \end{bmatrix},$$

which define the peak-to-peak output amplitude, γ_i, and frequency, ω_i, of each i^{th} unit oscillator in the CPG network. A more specific design of the matrix W, with respect to a gait frequency parameter, ω_s, is provided in (5.5).

5.2.1 Reflexive Gait Adaptations

Feedback signals are incorporated into the CPG through the frequency modulation matrix Ψ_ω and amplitude modulation matrix Ψ_M. These modulation parameters are generated using state estimates of the platform's orientation and angular rate. The platform orientation state-estimate, $\hat{\theta}_b$, is supplied to the controller from an Extended Kalman Filter utilizing inertial measurement and magnetometer feedback (which are separately calibrated). The platform's angular rate is output by an angular rate-gyro sensor.

Feedback-based corrections to the CPG aid in tracking a specified body orientation θ_b^r during gaiting. To achieve higher system velocities, it is often necessary to perform a gait wherein multiple legs leave contact with the ground. Configurations such as these would cause a quadruped robot to tip in the direction of the non-planted legs, thus disturbing θ_b. These disturbances are counteracted, in part, by adjusting the CPG such that the amount of torque applied on the main body by legs in flight is actively limited. This is done by adjusting stepping height and time-of-flight according to a measure of disturbance (essentially, the amount and rate of tipping). These adjustments have been formulated to mimic reflexive behaviors that might be performed by a biological system.

When the robot's body begins to fall in particular direction (*i.e.*, is disturbed by legs in-flight during gaiting), the disturbance signal

$$\dot{\epsilon}_\theta = R_{z_b}\left(\frac{\pi}{2}\right)\left(\dot{\hat{\theta}}_b - \dot{\theta}_b^r\right) \tag{5.2}$$

is non-zero. $\dot{\epsilon}_\theta$ represents a measure of translational drift recovered from gyroscopic sensor measurements. $R_{z_b}\frac{\pi}{2}$ represents a rotation by $\frac{\pi}{2}$ about the z-axis of the body frame O_b. $\dot{\epsilon}_\theta$ is mapped to the parameters Ψ_ω and Ψ_M by

$$\begin{aligned} \psi_i &= \text{sig}\left(w_i T_i - w_i c_i\right)\mu_i \\ \Psi_\omega &= I + A_\omega \text{diag}\left(\psi_i\right) \\ \Psi_M &= I - A_\mu \text{diag}\left(\psi_i\right) \end{aligned} \tag{5.3}$$

where

$$\begin{aligned} T_i &= \|\dot{\epsilon}_\theta\| \left(1 + \frac{\Delta x_i}{\|\Delta x_i\|}^T \frac{\dot{\epsilon}_\theta}{\|\dot{\epsilon}_\theta\|}\right) \\ \Delta x_i &= p_e - p_b \end{aligned} \tag{5.4}$$

with $\{w_i, c_i\} \in \Re$ and $\{A_\mu, A_\omega\} \in \Re^{4\times 4}$. sig $(*)$ represents the standard sigmoid step function. The signal T_i represents a projection of $\dot{\epsilon}_\theta$ into the unit-vector emanating from p_b to each i^{th} foot. This projection delegates the level of adjustment to each i^{th} oscillator as a result of $\dot{\epsilon}_\theta$.

Adjusting Ψ_ω by the method delineated in (5.3) serves to shorten the stepping period of a leg in-flight given greater values of $\dot{\epsilon}_\theta$. Likewise, Ψ_M is adjusted to decrease the height of each foot in-flight. The frequency of a full gaiting cycle is prescribed via ω_s and duty-factor $\alpha \in [0, 1]$ [16]. The matrix W is formed from ω_s and α as a follows:

$$W = \omega_s \alpha \begin{bmatrix} \frac{1}{1+e^{\zeta y_{2,1}}} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{1+e^{\zeta y_{2,4}}} \end{bmatrix} + \omega_s(1-\alpha) \begin{bmatrix} \frac{1}{1+e^{-\zeta y_{2,1}}} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{1+e^{-\zeta y_{2,4}}} \end{bmatrix} \tag{5.5}$$

and ζ being a sensitivity tuning parameter. A linear mapping between commanded platform velocity, v^r, and ω_s is prescribed such that stepping-cycle frequency is adjusted proportionally with respect to the desired velocity and turning rate of the system, as follows:

$$\omega_s = a_v v^r + a_\omega \|\omega^r\| \tag{5.6}$$

where a_v and a_ω are scalar gains. In BlueFoot's CPG implementation, the coupling matrix K takes on the values $k_{i,j} \in [-1, 1]$. Each element of the coupling matrix, $k_{i,j}$, is utilized to adjust the phase offsets between the unit-oscillators.Furthermore, setting $k_{i,j} = 1$ causes the j^{th} oscillator to attract the i^{th} oscillator towards a positive peak, and vice-versa. Setting $k_{i,j} = 0$ effectively disconnects i^{th} and j^{th} oscillators.

Figures 17 and 18 depict the CPG output, y_2, and corresponding unit-oscillator phase portraits of the oscillator dynamics, given by (5.1) with the feedback mechanism given in (5.3) used during simulations of BlueFoot's default two-pace trotting and four-pace walking gaits, respectively. The associated K matrix prescribing this gaiting pattern is a modified version of K for a 2-pace trot gait presented in [12], since this generates a more effective and fluid gait when applied to BlueFoot's gait controller

$$K \equiv \begin{bmatrix} 0 & -1 & 1 & -0.5 \\ -1 & 0 & -0.5 & 1 \\ -1 & -0.5 & 0 & -1 \\ -0.5 & 1 & -1 & 0 \end{bmatrix}. \tag{5.7}$$

A comparable four-pace *walking* gait can be achieved by a slight sign modifications on the elements of (5.7), which yields (5.8) as follows:

$$K \equiv \begin{bmatrix} 0 & -1 & 1 & -0.5 \\ -1 & 0 & -0.5 & 1 \\ -1 & 0.5 & 0 & -1 \\ 0.5 & -1 & -1 & 0 \end{bmatrix}. \tag{5.8}$$

Figure 19 shows the change in output state patterns during a transition from a four-paced to a two-paced gait. Figure 20 shows joint angle feedback from the actual robot platform during the execution of a CPG driven gait which achieves a ground-speed of $65\frac{mm}{s}$.

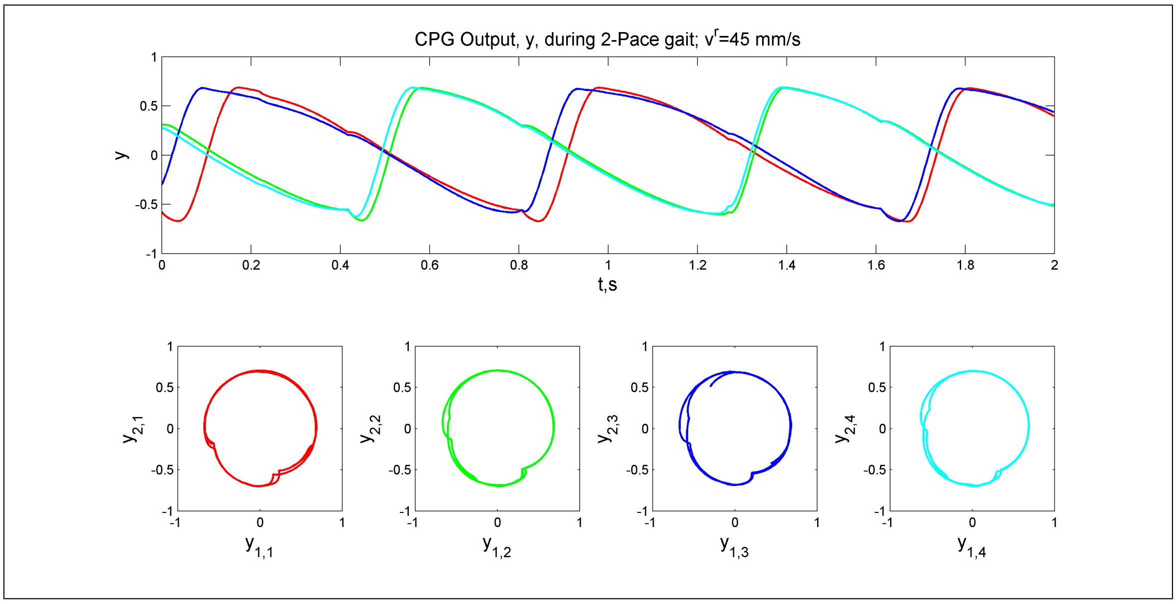


Figure 17: CPG output and phase plots for a two-paced gait over two gait cycles.

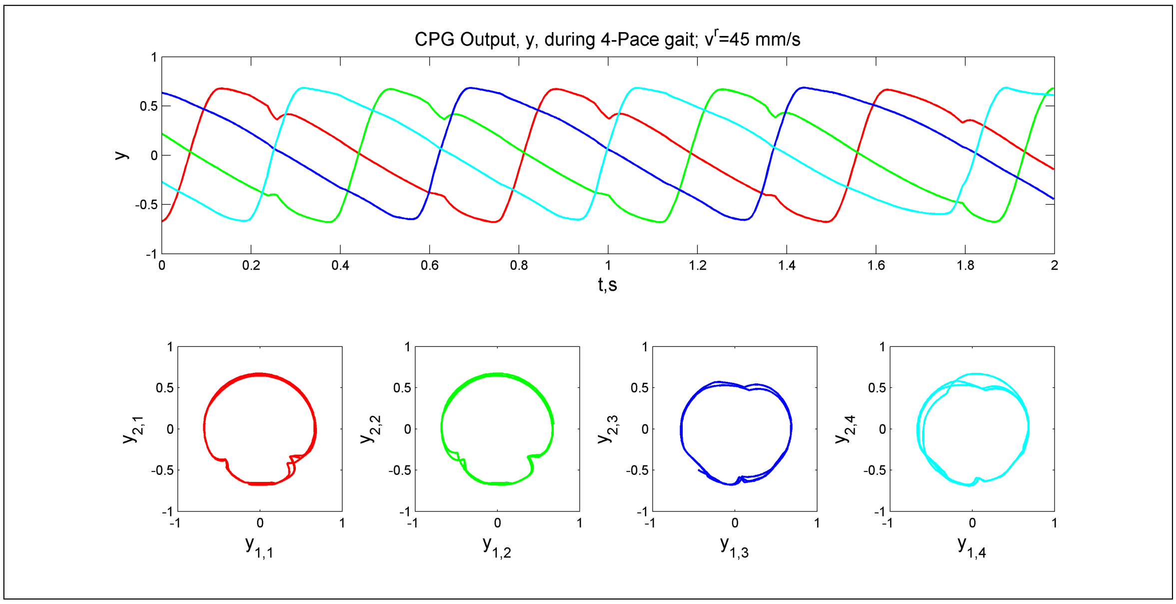


Figure 18: CPG output and phase plots for a two-paced gait over two gait cycles.

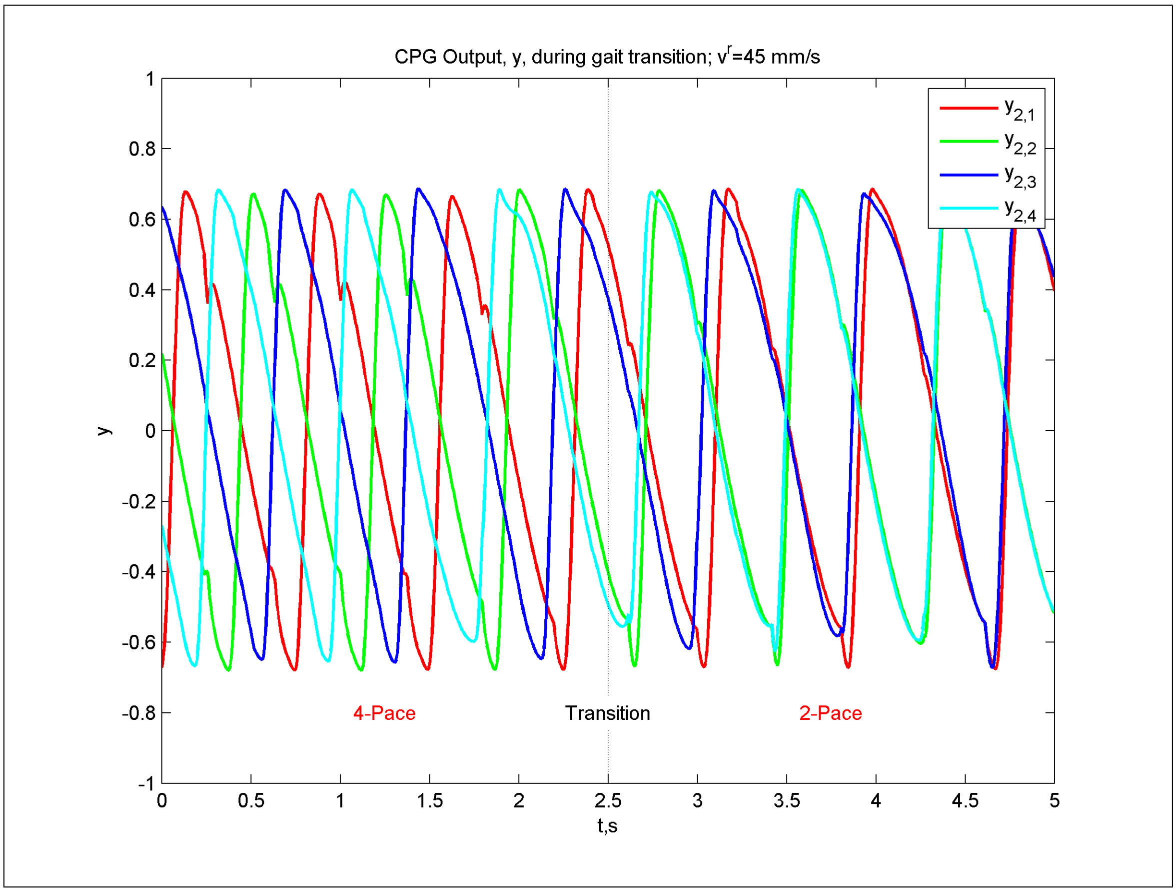


Figure 19: CPG output transition during four-pace to two-pace gait switch.

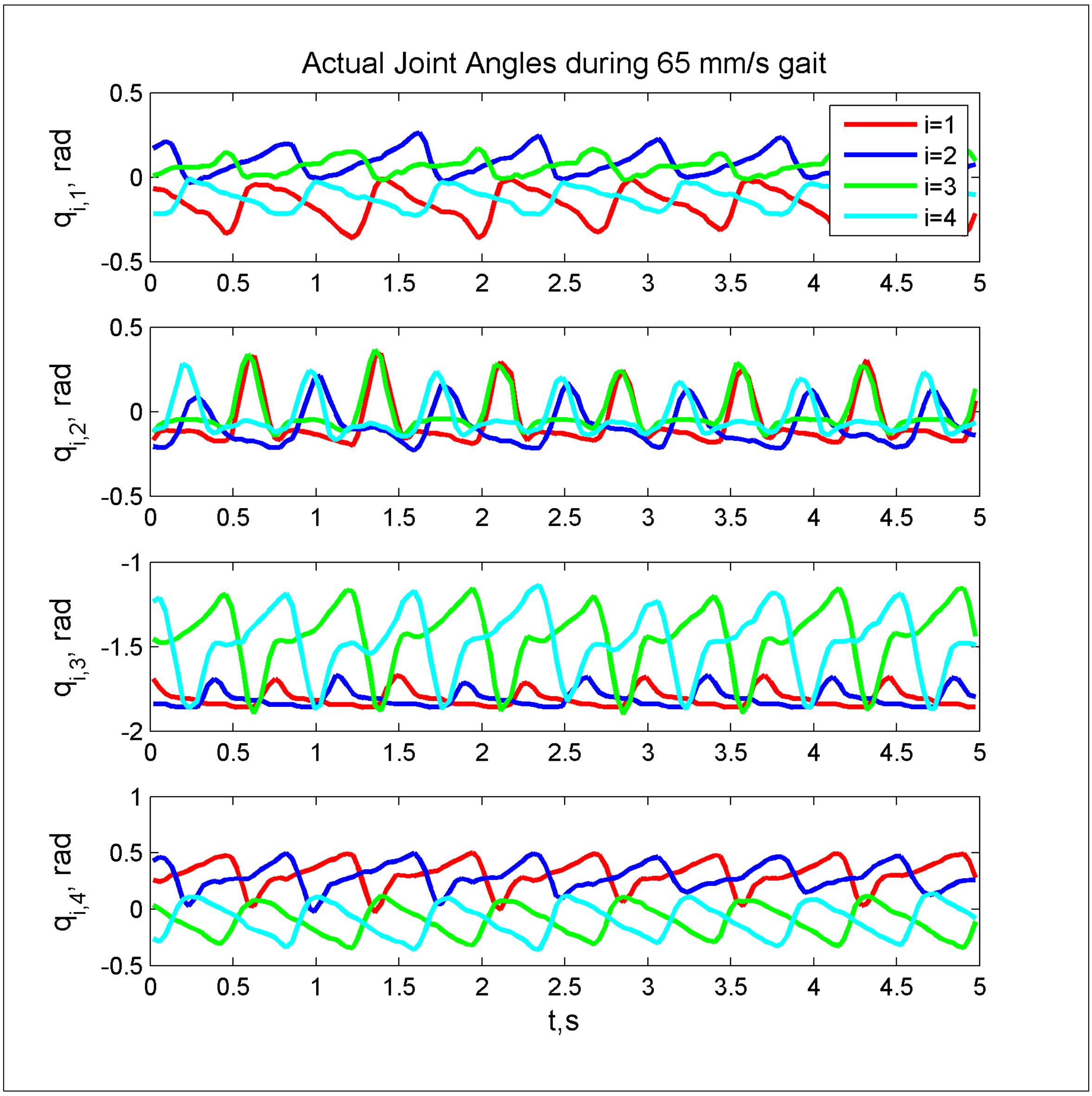


Figure 20: Real joint angles during a gait at $65\frac{mm}{s}$.

5.3 Foot Placement Control

A foot placement controller has been implemented which utilizes CPG outputs (defined in equation (5.1)) in concert with a virtual-force controller. This controller is designed such that each foothold of the robot tracks the foot positions of a virtual robot generated through a model reference. The virtual robot is described by the foot positions, $\tilde{p}_{i,e}$, a virtual reference configuration corresponding to the return-position, $\tilde{p}_c$, and orientation, $\tilde{\theta}_c$, of the main body in O_0. The robot follows the foot placement of the virtual robot to achieve a commanded ground speed $v^r > 0$ in a unit-vector direction

$\vec{u}^r \in \Re^3$; and turning velocity $\omega^r \in \Re$. Each virtual point is updated by:

$$\begin{aligned}
\dot{\tilde{\theta}}_c &= \omega^r \\
\dot{\tilde{p}}_c &= v^r \vec{u}^r \\
\dot{\tilde{p}}_{i,e} &= v^r \vec{u}^r + S\left(\omega^r \vec{h}_b\right) \tilde{R}_b \left(\tilde{p}_{i,e} - \tilde{p}_c\right)
\end{aligned} \tag{5.9}$$

where $\vec{h}_b$ is a unit vector that is orthogonal and pointed outward from the surface beneath the robot and $S\left(\omega^r \vec{h}_b\right) \in \Re^{3\times3}$ forms a skew-symmetric matrix from the vector argument $\omega^r \vec{h}_b$. The virtual foothold dynamics progress the target footholds at a commanded translational (forward) and rotational velocity, v^r and ω^r, respectively. The robot follows the virtual model at nearly the same velocities with minimal lag so long as system bandwidth is not exceeded. Using this virtual-foothold method is convenient as it allows for foot-placement planning to be independent of foot-trajectory planning. For example, terrain adaptation can be incorporated by modifying the location of virtual foot positions such that they conform to an upcoming surface. The robot's gait will then track these adaptations without any explicit modification of foot trajectories. In the event of contact with unperceived terrain, virtual-foothold positions will reset with respect to an estimated point of contact for each i^{th} contacting foot, *i.e.*, :

---

for all i **in** $i = \{1, ..., 4\}$ **do**
  if $y_{2,i} > 0$ **and** $\mu_i = 1$ **then**
    $\tilde{p}_{i,e} = \hat{p}_{i,e}$
  end if
end for

To clarify, $y_{2,i} > 0$ indicates that i^{th} foot is in flight, as per the definition of the foot-height controller to be defined in (5.12). The foot-position estimate $\hat{p}_{i,e}$ is defined in (4.10).

The full foothold controller represented by

$$\dot{p}_{i,e} = \dot{\tilde{p}}^v_{i,e} + \dot{\tilde{p}}^y_{i,e}, \;\; i = \{1, ..., 4\} \tag{5.10}$$

is a composite of a foot-repositioning and step-height controller which are formulated in terms of the dynamics of the signals $\dot{\tilde{p}}^v_{i,e}$ and $\dot{\tilde{p}}^y_{i,e}$, respectively. Each foot is treated as a point mass attracted to their corresponding virtual foothold by an over-damped mass-attractor system. $\dot{\tilde{p}}^v_{i,e}$ is defined by

$$\dot{\tilde{p}}^v_{i,e} = \frac{1}{m_i} \int^t P_{\vec{h}_i} \left(F_{s,i} - b_c \dot{p}_{i,e} + F_{\epsilon,i}\right) d\tau \tag{5.11}$$

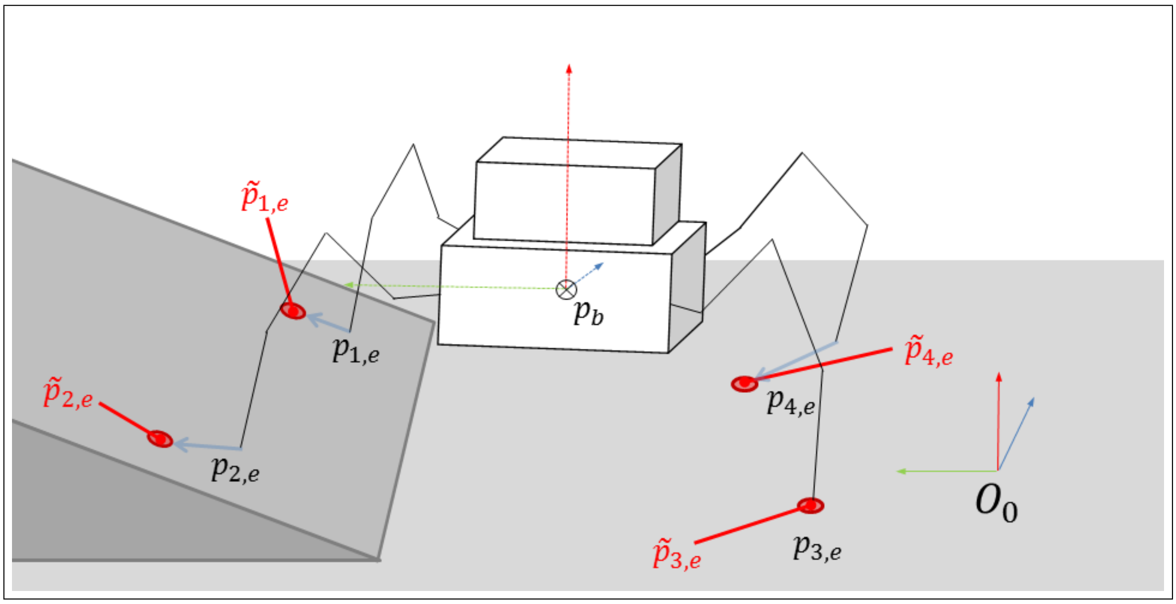


Figure 21: Virtual foothold representation. Blue arrows represent an attractive "force" between the feet and their corresponding virtual foothold.

where

$$
\begin{aligned}
F_{s,i} &= k_c \mu_i \left(\tilde{p}_i - p_{i,e}\right) U(y_{2,i}) \\
F_{\epsilon,i} &= a_\epsilon \dot{\epsilon}_\theta U(y_{2,i})
\end{aligned}
$$

with $k_c, b_c, a_\epsilon, m_i \in \Re$. k_c and b_c represent attraction and viscous damping constants for the virtual force system. $P_{\vec{h}_i}(*)$ projects the sum of forces, $(*)$, onto the surface beneath the i^{th} foot orthogonal to $\vec{h}_i$. $U(y_{2,i})$ is a standard unit step function. The force $F_{\epsilon,i}$ is a compensatory force to adjust for the inertial disturbance, $\dot{\epsilon}_\theta$. For instance, if the robot was pushed from the side of its body, this force would induce a side-stepping motion in attempt to catch the robot from falling in the direction of the push. Because of $U(y_{2,i})$, $F_{s,i}$ and $F_{\epsilon,i}$ are nonzero only when $y_{2,i}$ is positive.

The controller component $\dot{\tilde{p}}^y_{i,e}$ is defined from the CPG output signal $\dot{y}_{2,i}$. The height of each step is made proportional to the magnitude of desired platform velocity.

$$\dot{\tilde{p}}^y_{i,e} = \left(\alpha_v v^r g + \alpha_\omega \left\|\omega^r\right\|\right) \dot{y}_{2,i} \vec{h}_i \tag{5.12}$$

where α_v and α_ω are scalar weighting parameters. Scaling step height relative to command parameters v^r and ω^r has been seen in experimental studies to be more effective than using a fixed step-height for all gait configurations. Figures 22 and 23 show stepping patterns generated using the aggregate virtual-force foothold controller and CPG motion outputs during two and four-paced gaits, respectively, which achieve a forward velocity of $v^r = 40\frac{mm}{s}$. The sequence of foot contacts made during each gait is enumerated in each figure. Red, green, blue, and cyan dots represent contacts made by the front-right, front-left, back-left and back-right feet, respectively.

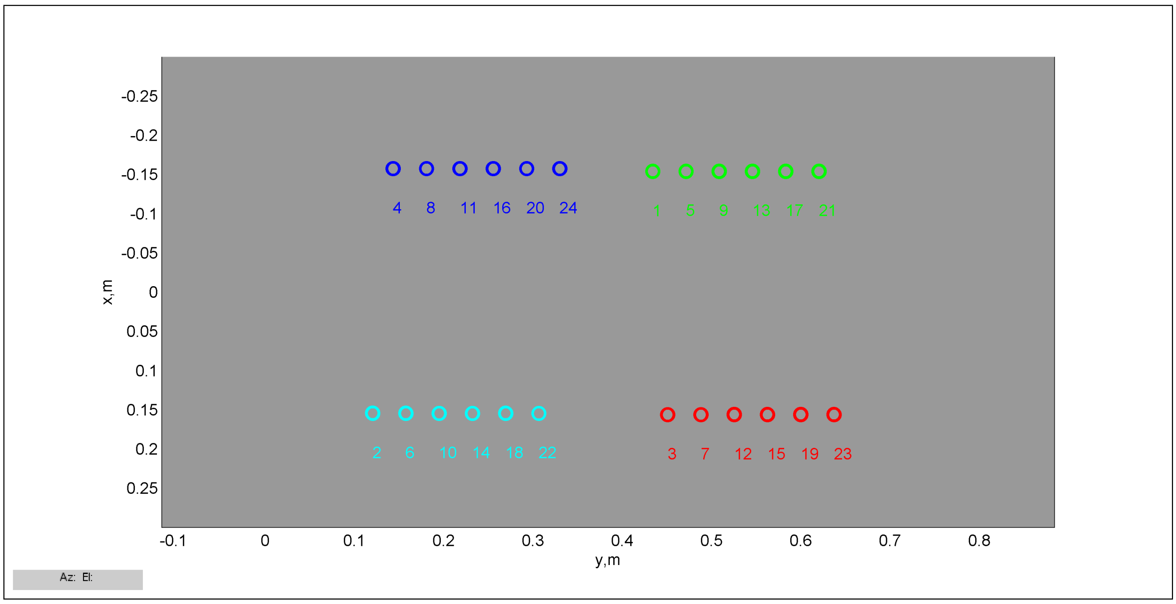


Figure 22: Stepping pattern generated using virtual-force foothold controller during two-pace gait sequence at $v^r = 40\frac{mm}{s}$.

5.4 ZMP Based Trunk-Placement Control

A modified Zero-Moment-Point (ZMP) based controller is utilized in positioning BlueFoot's body during gaiting. In this context, the ZMP of the system is a set of state values for which the net-torque exerted upon the system, about the COG, is zero [59,60]. Unlike [61,62], which address ZMP-based control by considering the robot's body as point-mass with massless limbs, this method takes into account the torque contribution of each non-supporting leg. Each leg in flight is considered as a series of point masses whose locations are computed using joint position feedback and trunk orientation estimates, in concert with robot's forward kinematic model.

The ZMP approach is used to first calculate static ZMP configurations (*i.e.*, $\ddot{p}_{COG} \approx 0$) at each time instance. The distance between the robot's COG and associated ZMP is incrementally minimized on-line by treating the ZMP as a mass-attractor and "pulling" a reference body position, $p_b^{b',r}$, towards the ZMP point at each update. This differs from

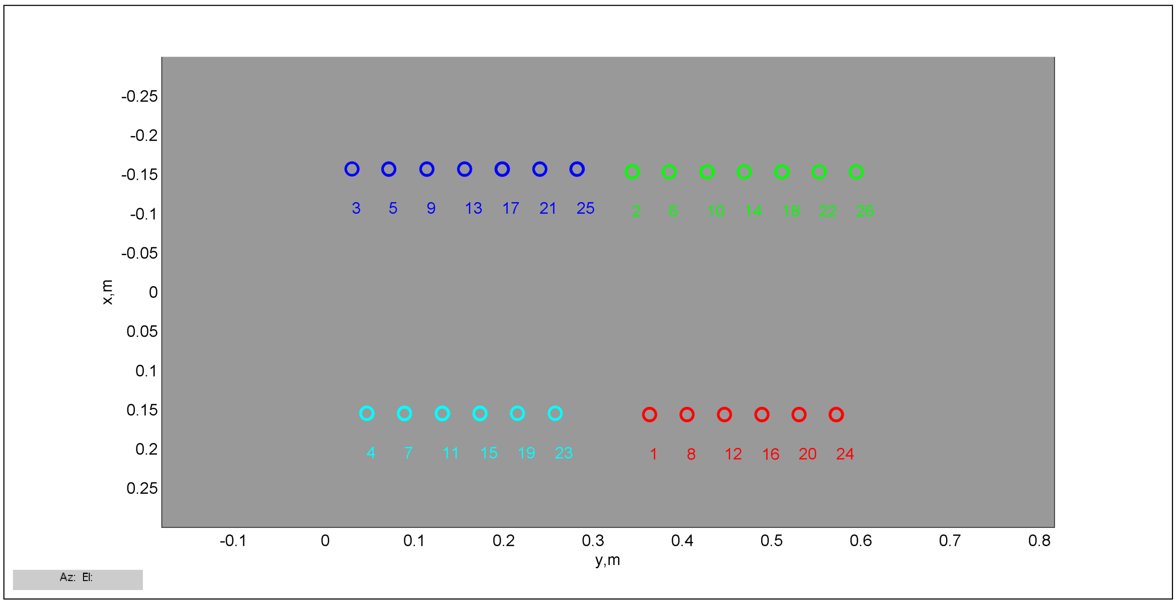


Figure 23: Stepping pattern generated using virtual-force foothold controller during four-pace gait sequence at $v^r = 40\frac{mm}{s}$.

approaches similar to [62] because these routines feature off-line, trunk trajectory design which aim to minimize deviations between the robot's COG and ZMP by creating an appropriate limit-cycle motion for the robot's trunk. These approaches typically utilize simplified, linearized models of the actual system about marginally-stable equilibrium points. To realize an adjustment of the robot's COG, towards the ZMP, BlueFoot's trunk position is modified during the execution of a gait. The trunk is controlled (and not individual feet) as it contributes most of the system's total mass and, thus, has the greatest influence over the location of the platform's COG. Restricting adjustments to the trunks translational states allow pre-planned foot trajectories to remain unmodified by the ZMP-control module during gait execution, thus decoupling foot-placement and body-placement control.

To incrementally compute the static ZMP of the platform, a measure of net-moment on the body of the platform must be known. The net-moment due to gravitational forces, τ_{net}, is approximated using the locations of each link and joints as point-mass loads. Hence, τ_{net} is calculated as follows:

$$\tau_{net} = \left(\vec{p}_b^{\,b'} - \hat{p}_{COG}^{\,b'}\right) \times (\vec{g} m_b) + \tau_{legs} \tag{5.13}$$

where

$$\begin{aligned}
\tau_{legs} &= \sum_{i=1}^{4}(1-\mu_i)\left(m_{i,e}p_{i,e}^{b'} + \sum_{i=1}^{4}\left(m_{i,j}^{J}d_{i,j}^{J} + m_{i,j}^{L}d_{i,j}^{L}\right)\right)\times\vec{g} \\
d_{i,j}^{J} &= p_{i,j}^{b'} - \hat{p}_{COG}^{b'}. \\
d_{i,j}^{L} &= \begin{cases} 0.5\left(p_{i,j+1}^{b'} - p_{i,j}^{b'}\right) + p_{i,j}^{b'} - \hat{p}_{COG}^{b'} & \text{if } j<4 \\ 0.5\left(p_{i,e}^{b'} - p_{i,j}^{b'}\right) + p_{i,j}^{b'} - \hat{p}_{COG}^{b'} & \text{if } j=4 \end{cases}
\end{aligned}$$

with m_b, $m_{i,j}^{J}$, $m_{i,j}^{L}$, $m_{i,e}$ represent the mass of the main body; the mass of each joint; the mass of each link; the mass of each foot, respectively. μ_i is included in the above formulation such that only stepping (non-contacting) legs contribute to τ_{legs}. It should be noted that the superscript b' denotes that each particle considered in this controller is defined with respect to the robot-relative frame $O_{b'}$ defined in Section 4.1. This allows the control routine to be performed without explicit knowledge of robot's absolute position in O_0. The estimated center of gravity in $O_{b'}$, $\hat{p}_{COG}^{b'}$, is generated as follows:

$$\begin{aligned}
\hat{p}_{COG}^{b'} &= \frac{1}{m_T}\left(m_b p_b^{b'} + \sum_{i=1}^{4}\left(m_{i,e}p_{i,e}^{b'} + \sum_{j=1}^{4}\left(m_{i,j}^{J}p_{i,j}^{b'} + m_{i,j}^{L}p_{i,j}^{b',L}\right)\right)\right) \\
p_{i,j}^{L} &= \begin{cases} 0.5\left(p_{i,j+1}^{b'} - p_{i,j}^{b'}\right) + p_{i,j}^{b'} & \text{if } j<4 \\ 0.5\left(p_{i,e}^{b'} - p_{i,j}^{b'}\right) + p_{i,j}^{b'} & \text{if } j=4 \end{cases}
\end{aligned}$$

where

$$m_T = m_b + \sum_{i=1}^{4}\left(m_{i,e} + \sum_{j=1}^{4}\left(m_{i,j}^{J} + m_{i,j}^{L}\right)\right) \tag{5.14}$$

Here, the the local COG of each linkage, $p_{i,j}^{L}$, is assumed to be at the midpoint along the length of each link, half-way between two successive joints. Setting $\tau_{net}=0$, (5.13) is manipulated to a derived solution for the ZMP as follows:

$$p_{ZMP}^{b'} = R_{z_P}\left(\frac{\pi}{2}\right)\left(\frac{\|g\|}{m_b}\right)\tau_{legs} + \hat{p}_{COG}^{b'}. \tag{5.15}$$

Using $p_{ZMP}^{b'}$, the platform's posture is then updated using a virtual-force controller. Moreover, $p_b^{b'}$ is to be controlled through p_b^r such that it is smoothly attracted to p_{ZMP}. The controller is given by

$$\dot{p}_b^r = P_{\vec{h}_i}\left(K_Z(p_{ZMP}^{b'} - p_b^{b'}) + K_F\frac{F_r}{m_b}\right) \tag{5.16}$$

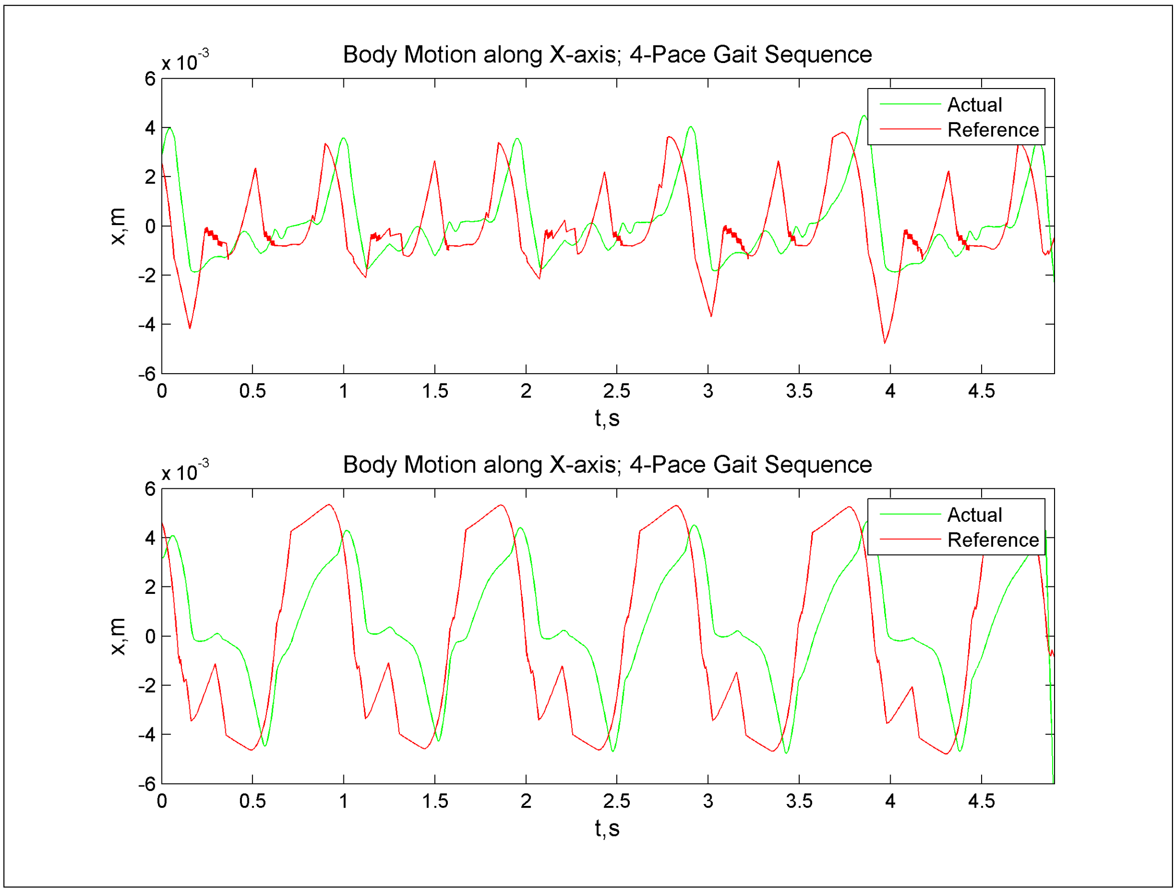


Figure 24: Body motion generated using ZMP controller during two-pace *(top)* and four-pace *(bottom)* gaiting sequence.

where

$$\begin{aligned} F_r &= \sum_{i=1}^{4} e^{\left(k_l r_i^-\right)} + e^{\left(k_l r_i^+\right)} \\ r_i^+ &= \left\| p_{i,e}^{b'} - p_b^{b'} \right\| - r_{max} \\ r_i^- &= r_{min} - \left\| p_{i,e}^{b'} - p_b^{b'} \right\| \end{aligned}$$

and $K_Z, K_F, K_c, K_\epsilon, k_l > 0$ and $\dot{p}_b^{b',r}$ is the reference body velocity. F_r is a boundary force added to ensure that the workspace of each manipulator, defined by the radii r_{min} and r_{max}, is not exceeded when the body is repositioned. k_l is picked to be adequately large such that this force is nearly zero when the body and foot positions comply with the local workspace of each leg, and large when the workspace is nearly compromised, forcing the placement of the body to comply with each leg workspace. Figure 24 shows a trajectory for the trunk position reference, $p_b^{b',r}$, and actual body trajectory, $p_b^{b'}$ generated using the aforementioned ZMP-based body-placement controller during a gait which achieves a forward velocity of $v^r = 40\frac{mm}{s}$. Here, results are shown for the x-axis motion of the body. This is because during forward motion, the most significant alterations to the location of the body are seen along the x-axis.

5.5 Trunk Leveling via NARX-Network Learning Approach

Suppressing disturbances which enter a quadruped system during gaiting is a matter which requires special handling, given the the general dynamical complexity of both the robot and its interactions with the its environment. This complexity can be handled, in part, by a learning-based approach. This section will focus on the formulation of a learning controller used to reject disturbances from the orientation states of the trunk of a legged robot. Disturbance rejection from the trunk sub-system of a legged platform has practical significance when carrying a payload (such as cameras, optical systems, armaments, etc.) rigidly fixed to their main body. Disturbances are imparted upon the trunk during gaiting in two main ways:

1. instantaneous changes in force distribution when feet make and break contact with the ground

2. under-actuation that occurs during certain dynamic gaits. During dynamic gaits, such as trot gaits, the state of contact between the feet and the ground is changed often so as to prevent the walking robot from tipping past a recoverable configuration.

Additionally, these gaits feature the utilization of two or fewer legs to support the trunk at any given time, causing the system to enter an under-actuated state where the body is free to rock about the planted feet, as shown in Figure 25. To achieve disturbance rejection on the trunk orientation and to attain a fixed orientation, experimentation has been performed using a control methodology which utilizes a Nonlinear Autoregressive Neural Network with Exogenous inputs (NARX-NN) as part of an active compensation mechanism. The network is used to estimate the system dynamics and, further, predict periodic disturbances in an on-line fashion. The compensator is utilized to modify referential joint trajectories by way of a weighted sum between the original joint trajectories generated by the gaiting mechanism and a reference correction signal generated by the compensator.

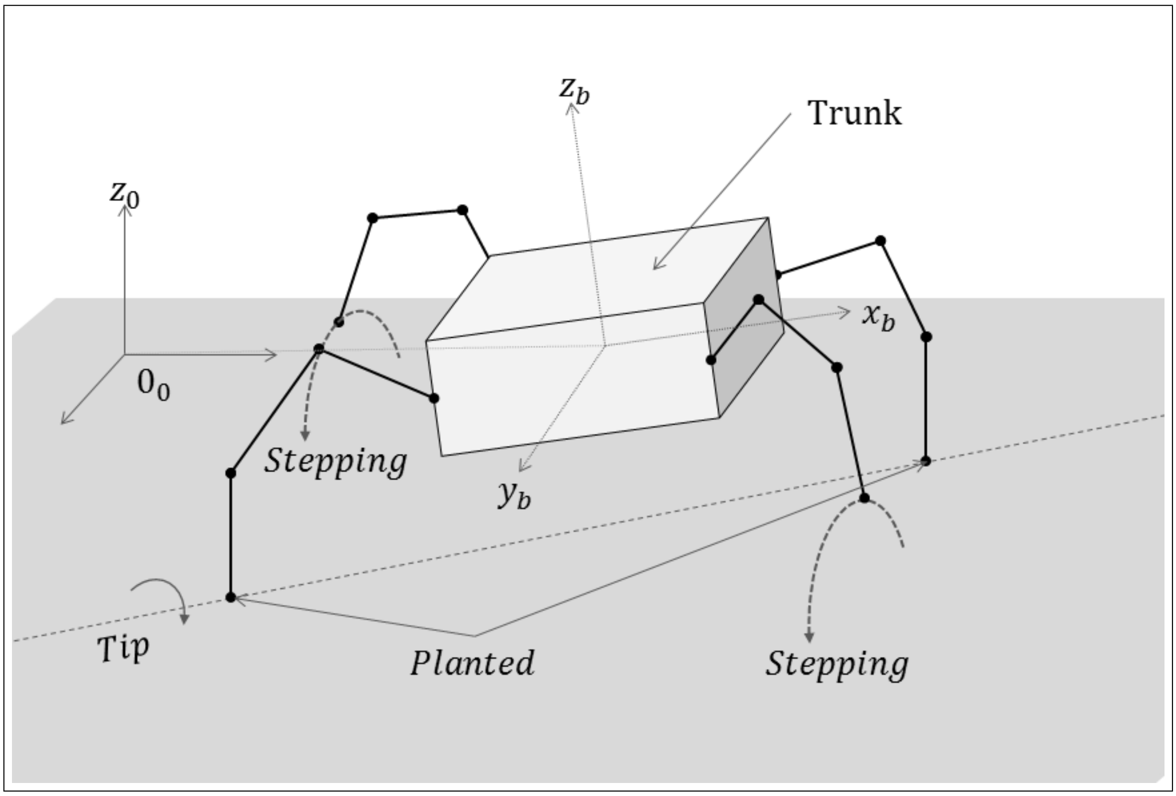


Figure 25: Quadruped tipping about planted feet.

5.5.1 NARX-Neural Network

Using a NARX-NN (introduced in Chapter I), the dynamical estimate $\hat{\Phi}_k$ is generated as a prediction of the sampled system dynamics, $\hat{\psi}_{k+1}$. The relationship between $\hat{\Phi}_k$ and the network prediction $\hat{\psi}_{k+1}$will be made clear in the description of the network training signal given in (5.18). The general input-output relationship of the NARX-NN predictor, $\mathcal{N}$, is described as follows:

$$\begin{aligned} \hat{\psi}_{k+1} &= \mathcal{N}(\hat{\Psi}_k^N, U_k^N) \\ \hat{\Psi}_k^N &= [\hat{\psi}_k, \hat{\psi}_{k-1}, ..., \hat{\psi}_{k-N+1}] \\ U_k^N &= [u_k, u_{k-1}, ..., u_{k-N+1}] \end{aligned} \tag{5.17}$$

where U_k^N and $\hat{\Psi}_k^N$ are collections of N most recent samples of the network inputs, u_k, and the network output, $\hat{\psi}_k$, respectively. The NARX-NN input, u_k, represents a tuple $u_k = (z_{1,k}, z_{2,k}, f_{ext,k})$ whose components are the arguments of Φ at time instant k.

5.5.2 NARX-NN Training Regimen

The NARX-NN training signal is formulated to estimate Φ_k from the system dynamics. By (4.21), it can be seen that Φ_k can be estimated if $z_{2,k+1}$ can be predicted. We consider the following target network prediction output, ψ_{k+1}, defined by:

$$\psi_{k+1} = \tau_k - \hat{M}_{1,k}(z_{2,k+1} - z_{2,k})\Delta_s^{-1} = \Phi_k - e_{2,k}^{\Delta_s} \ . \tag{5.18}$$

This training signal formulation assumes that $\hat{M}_{1,k}$ represents $M_{1,k}$ exactly, which is likely not the case given the system's complexity. In the absence of a well-modeled $\hat{M}_{1,k}$, a constant symmetric $\hat{M}_{nom}$ will be picked such that $\hat{M}_{1,k} = \hat{M}_{nom} \forall k$. $\hat{M}_{nom}$ has the following structure:

$$\hat{M}_{nom} = \begin{bmatrix} \hat{M}_{bb} & \hat{M}_{bq} \\ \hat{M}_{qb} & \hat{M}_{qq} \end{bmatrix} \tag{5.19}$$

where $\hat{M}_{bb} \in R^{6\times 6}$, $\hat{M}_{bq} = \hat{M}_{qb}^T \in R^{6\times 16}$, and $\hat{M}_{qq} \in R^{16\times 16}$. It is particularly important that $\hat{M}_{bq} \neq 0$ to reflect some degree of coupling between the joint states q and the trunk states p_b and θ_b. In general, if $\hat{M}_{nom}$ should be selected to reflect the *average* system mass matrix over the range of configurations, z_1, seen during gaiting. This approximation has shown to be adequate from our results, and depends on the assumption that changes in $\hat{M}_{1,k}$ are small over the subset of state z_1 experiences during a periodic gaiting sequence. Future improvements of this controller involve the formulation of a separate estimator for $M(z_1)$, or a control / learning scheme with no direct dependence on $M(z_1)$.

Since $\hat{\psi}_{k+1}$ is non-causal, training is performed one time-step after a prediction is made using the input-output pair $\hat{\psi}_k$ and $\{\Psi_{k-1}^N, U_{k-1}^N\}$. Note that $\hat{\psi}_k$ can calculated directly using (5.18) where all component signals are time-delayed by one time-step. Training can then be described by:

$$\psi_k \xrightarrow{BP(\gamma^{lr})} \mathcal{N}(\Psi_{k-1}^N, U_{k-1}^N) \tag{5.20}$$

where $\gamma_{min}^{lr} < \gamma^{lr} < \gamma_{max}^{lr}$ is a learning rate adapted using a *bold-driver* update routine. Bold-driver learning-rate adaptation is a heuristic method for speeding up the rate of convergence of back-propagation training regimes [63, 64]. This γ^{lr} update law is parameterized by $\beta \in (0, 1)$ and $\zeta \in (0, 1)$ which are selected to specify the amount by which γ^{lr} increases or decreases per update, and γ_{min}^{lr} and γ_{max}^{lr} which are used to saturate γ^{lr}. The bold-driver scheme utilizes the current and previous mean-squared network output error values (MSE_k and MSE_{k-1}, respectively) to adjust γ^{lr} as follows:

$$\gamma^{lr} \leftarrow \begin{cases} \gamma^{lr}(1-\beta) & \text{if } MSE_k > MSE_{k-1} \\ \gamma^{lr}(1+\zeta\beta), & \text{otherwise.} \end{cases} \tag{5.21}$$

Since network training is being performed on-line as an incremental routine, the effective mean-squared NARX-NN output error is low-passed by a factor $\lambda \in (0,1)$. This update technique has been selected to ensure that outliers presented during training do not affect network learning updates as significantly as "nominal" training pairs. Network output error, $e_{\mathcal{N},k}$, and its associated MSE values are calculated after each prediction by:

$$\begin{aligned} e_{\mathcal{N},k} &= \hat{\psi}_k - \psi_k \\ MSE_k &\leftarrow \lambda \|e_{\mathcal{N},k}\|_2^2 + MSE_{k-1}(1-\lambda). \end{aligned} \tag{5.22}$$

5.5.3 Compensator Output

The control scheme is first presented with respect to the servo input torques, $\tau_{q,k}$, and formulated to achieve a level trunk characterized by $\theta_b = 0$, $\dot{\theta}_b = 0$. To formulate this controller, the dynamical sub-system which corresponds to the un-actuated trunk orientation states is isolated by:

$$\ddot{\theta}_b = \Gamma_1 M^{-1}(z_1)(\Gamma_2 \tau_q + \Phi) \tag{5.23}$$

where

$$\begin{aligned} \Gamma_1 &= [0_{3\times3}, I_{3\times3}, 0_{3\times16}] \\ \Gamma_2 &= [0_{16\times6}, I_{16\times16}]^T \end{aligned}$$

and $\Gamma_2\tau_q$ is equivalent to the original system input, τ. In order to enforce a level platform with zero angular velocity, we seek a τ_q which emulates the proportional-derivative (P.D.) control law:

$$\ddot{\theta}_b = -K_b\theta_b - K_d\dot{\theta}_b \tag{5.24}$$

where K_b and K_d are constant gain matrices. Using this P.D. law and (5.23), we propose a least-squares solution for τ_q by:

$$\tau_q \approx -\left[\Gamma_1 M^{-1}(z_1)\Gamma_2\right]^{\dagger} \Gamma_1 M^{-1}(z_1)(\Phi + K_b\theta_b + K_d\dot{\theta}_b) \tag{5.25}$$

where $[*]^\dagger$ denotes the Penrose-Moore pseudo-inverse of $[*]$. Replacing all dynamical terms with their associated discrete-time equivalents, and Φ by the NARX-NN output $\hat{\Phi}_k = \hat{\psi}_{k+1}$, we apply (5.25) to arrive at the following required joint torque estimate:

$$\hat{\tau}_{q,k} = -\left[\Gamma_1 \hat{M}_{1,k}^{-1} \Gamma_2\right]^\dagger \Gamma_1 \hat{M}_{1,k}^{-1} (\hat{\psi}_{k+1} + K_b \theta_{b,k} + K_d \dot{\theta}_{b,k}) \tag{5.26}$$

where $\theta_{b,k}$ and $\dot{\theta}_{b,k}$ are samples of angular trunk position and rate, respectively.

Using the joint controller dynamics presented in (4.22) and the estimate $\hat{\tau}_{q,k}$, we can formulate a reference-trajectory correction which is used to alter the joint reference positions, q_k^r. Moreover, the corrected reference position, $q_{1,k}^{r,*}$ is generated such that the estimated output torque $\hat{\tau}_{q,k}$ is attained by each joint controller. This joint-reference compensator output is defined using (5.26) as follows:

$$q_{1,k}^{r,*} = k_s^{-1}\left(\hat{\tau}_{q,k}\right) + q_{1,k}. \tag{5.27}$$

The correction signal, $q_k^{r,*}$, is combined with the original gaiting trajectory signal, q_k^r, as a weighted sum to form a compensated joint control reference signal, $\tilde{q}_k^r$, defined by:

$$\tilde{q}_k^r \leftarrow (1-\alpha) q_k^r + \alpha(q_k^{r,*}) \tag{5.28}$$

where $\alpha \in (0,1)$ is a uniform mixing parameter. The parameter α must be tuned with respect to the stability margins of the gait being compensated. The resultant $\tilde{q}_k^r$ is then applied to each joint controller in place of the original reference signal, q_k^r, generated by the gait controller. Selection of the parameter α is crucial for achieving good performance.

5.5.4 NARX-NN Compensator Results

The NARX-NN compensator has been tested, exclusively, in simulation and has been applied to applied to the quadruped as it executes a stable CPG-driven trot gait depicted in Figure 25. In these trials, gaiting frequency is adjusted accordingly to achieve particular forward speeds. NARX-NN parameters are fixed for all trials with learning-rate parameters set to $\beta = 0.0001$, $\zeta = 0.0005$ and $\lambda = 0.01$. The NARX-Network is configured with two hidden layers containing 50 neurons each. Each input and hidden-layer neuron is modeled using a symmetric sigmoid activation function. Output layer neurons are modeled using linear activation functions to avoid output-scaling saturation issues. Figure 27 exemplifies the convergence of the NARX-NN prediction error when the platform executes a gait at 100 $\frac{mm}{s}$ with $\alpha = 0.35$.

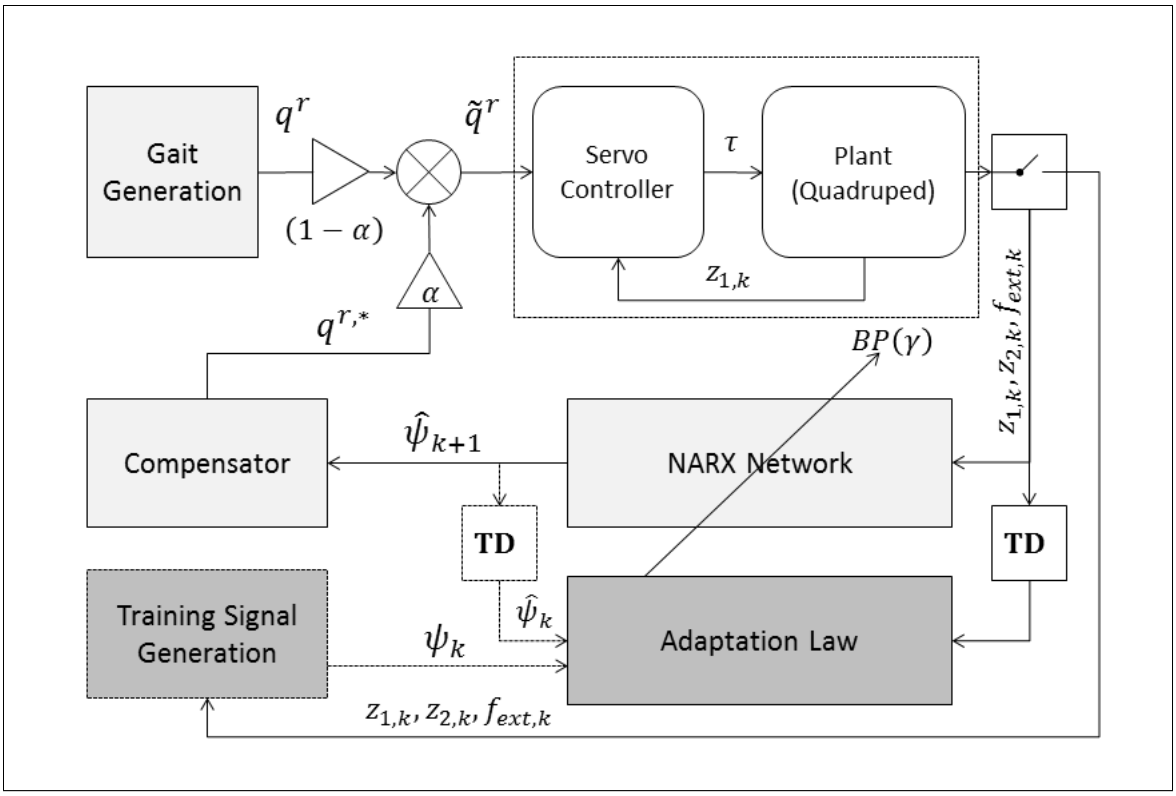


Figure 26: Full system diagram with NARX-NN compensator mechanism.

Contact forces, $f_{i,ext}$, are handled in the simulated implementation as the would be on the real BlueFoot platform. This handling method accounts for the absence of true force-torque sensors on each of BlueFoot's feet and, instead, makes use of the system's binary foot-contact sensors, in conjunction with the IMU. Although contact forces on each foot are accessible in simulation, the force at each foot is estimated using a combination of trunk 3-axis accelerometers and foot contact data. Assuming a rigid system and a uniform distribution of forces to each planted foot, a rough estimate of the force applied to each i^{th} planted foot, $\hat{f}_i$, can be generated by:

$$\hat{f}_i = m_T \mu_i \left(\ddot{p}_b - \vec{g}\right) / \sum_{j=1}^{4} \mu_j \tag{5.29}$$

where m_T represents the total system mass; $\mu_i \in \{0, 1\}$ is the contact state of the i^{th} foot (a value $\mu_i = 1$ represents contact); $\vec{g}$ is the gravity vector; and $\ddot{p}_b$ is the trunk acceleration in the world frame. Ideally, the measurement of f_i would be obtained via a 3-axis force-torque sensor placed at each foot.

All simulated trials are performed over a period of 60 seconds each. During the first 10 seconds of each simulation, the robot moves from sitting position to a standing position and initiates walking. During each simulation period, the NARX-NN compensator is activated (not training) and deactivated (training) every 10 seconds. Figures 28, 29 and 30 depict an initial set of simulation results showing the effect of varying the mixing parameter $\alpha \in \{0.125, 0.25, 0.35\}$. For all such trials, the robot performs a trot-gait which achieves a forward speed of 60 $\frac{mm}{s}$. It is expected that as α increases, the compensator will have greater authority over trunk stabilization. From these results, we observe that for all α, disturbance magnitude is decreased to some extent. However, for smaller α, the compensator is less effectual due to the fact that it has less authority over joint reference signals. From the results in Figure 29, it is shown that the compensator improves pitch stability by more than roughly 50% and roll stability by more than 60%. Figure 31 shows the compensator's performance at higher gaiting speeds of 80 $\frac{mm}{s}$ and 100 $\frac{mm}{s}$. Here the controller improves both pitch and roll by nearly 50% and 40% of the uncompensated signal magnitude, respectively.

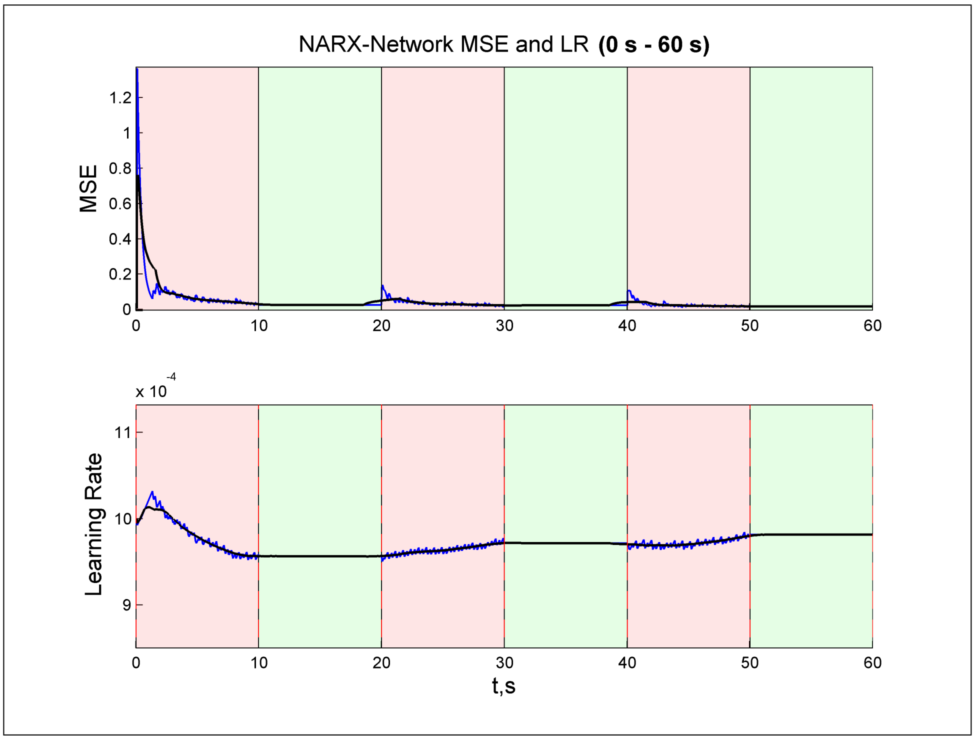


Figure 27: NARX Network MSE convergence for trial shown in Figure 32

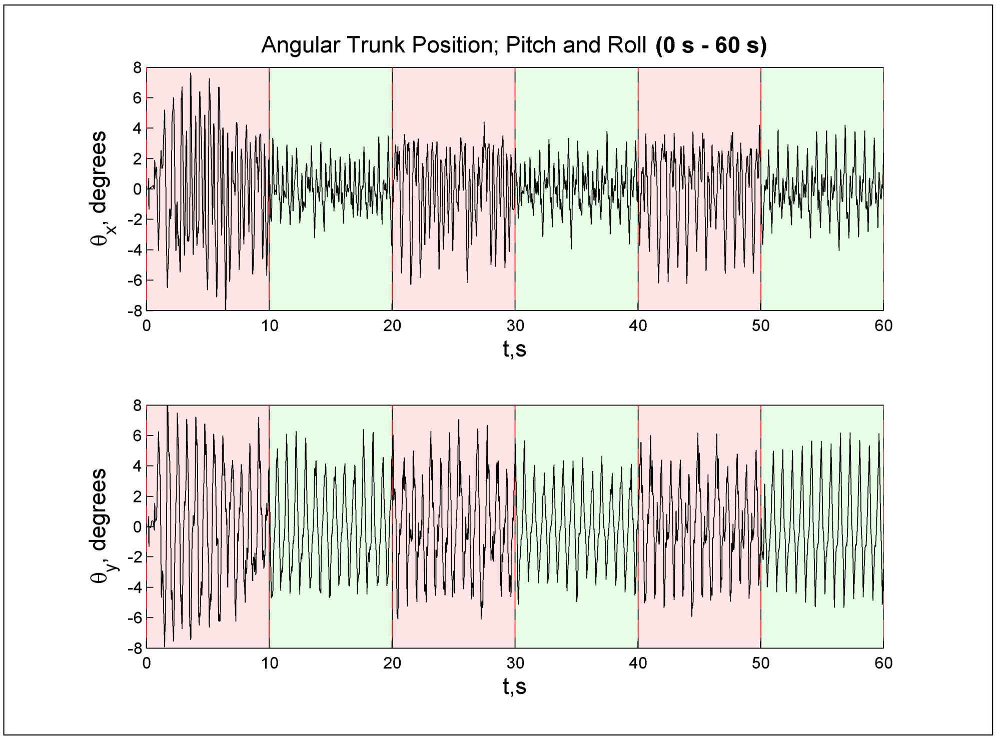


Figure 28: Trunk orientation during 60 $\frac{mm}{s}$ gait with $\alpha = 0.125$.

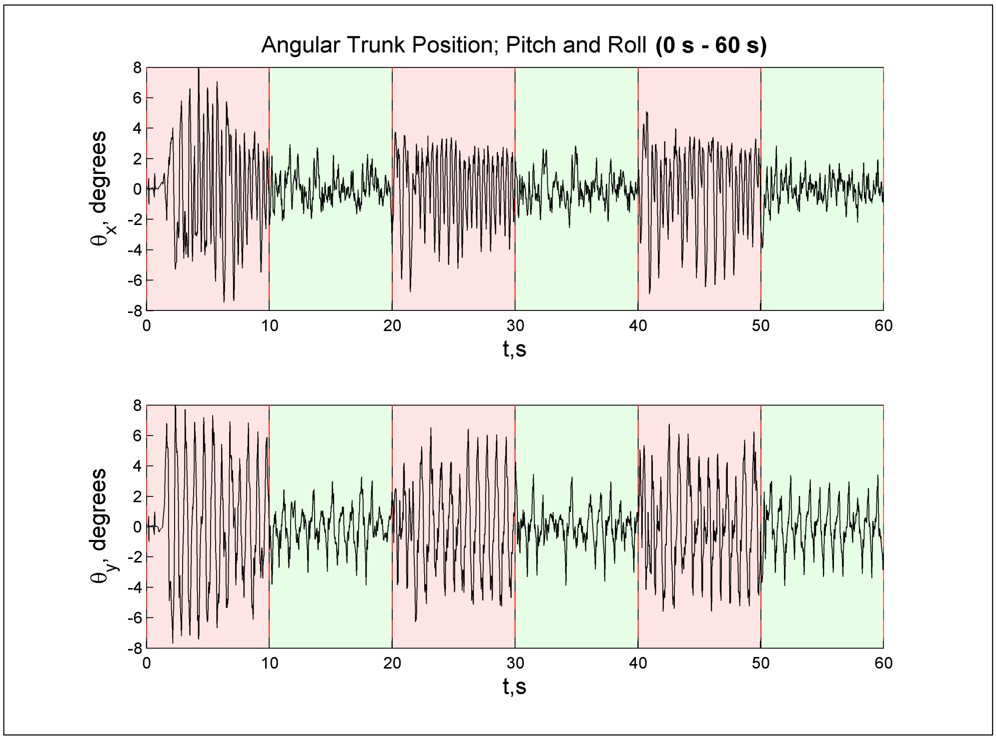


Figure 29: Trunk orientation during 60 $\frac{mm}{s}$ gait with $\alpha = 0.250$.

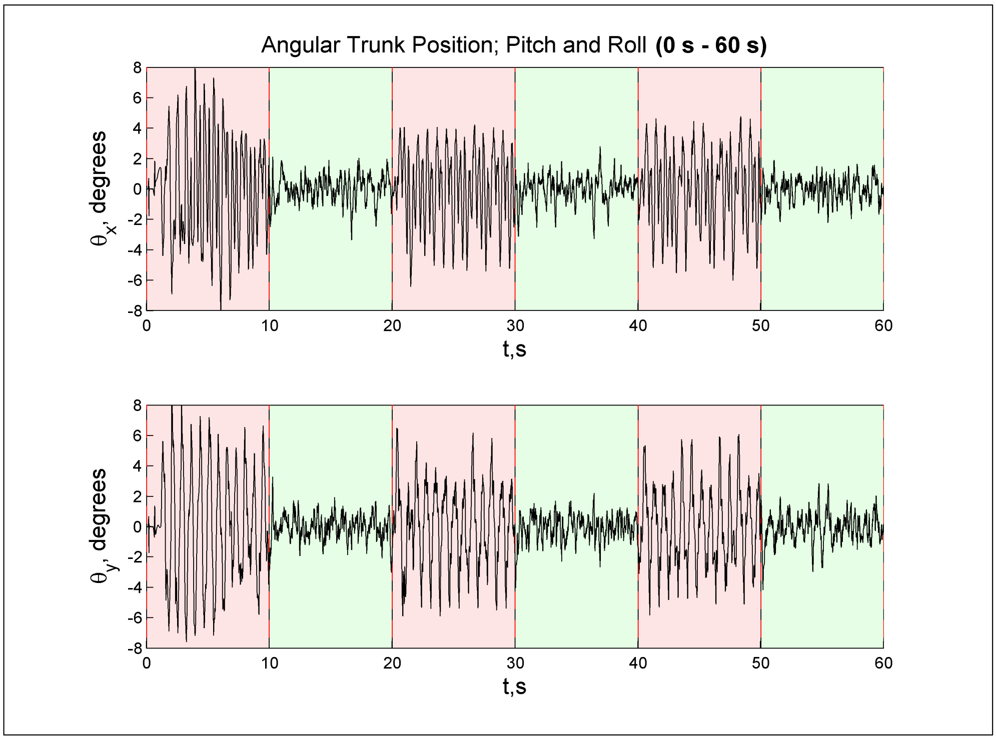


Figure 30: Trunk orientation during 60 $\frac{mm}{s}$ gait with $\alpha = 0.350$.

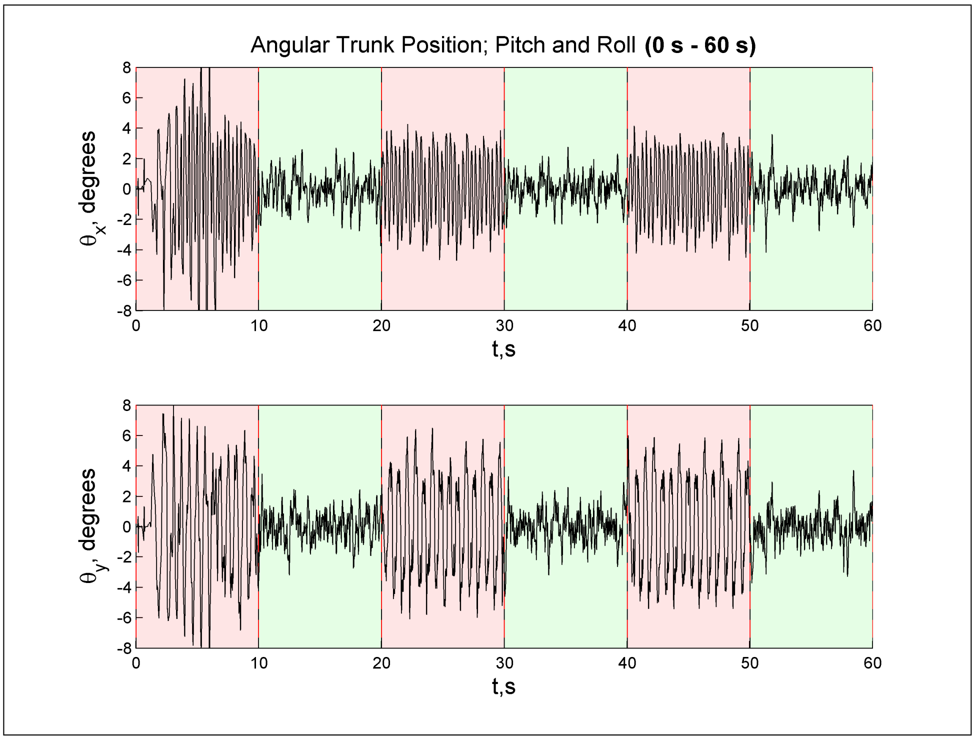


Figure 31: Trunk orientation during 80 $\frac{mm}{s}$ gait with mixing parameter set to $\alpha = 0.35$

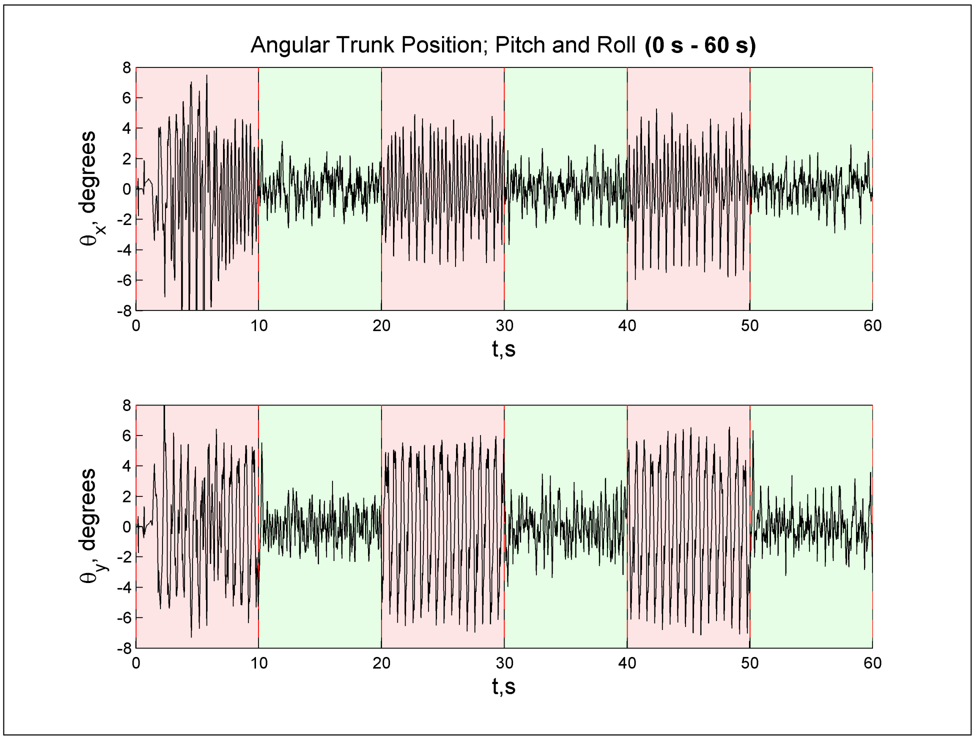


Figure 32: Trunk orientation during 100 $\frac{mm}{s}$ gait with $\alpha = 0.35$

CHAPTER VI

Perception and Navigation

6.1 Flatland navigation and goal (feature) tracking

The BlueFoot robot is operated using a hybrid, potential-field control scheme for obstacle avoidance during navigation over flatland. Camera-based feature tracking has been incorporated into this navigation scheme to allow for dynamic object tracking and aid in the aversion of local minima. This hybrid navigation routine administers commands to the robot's gaiting controller using two main parameters, v^r and ω^r, which represent forward velocity and turning rate of the platform with respect to the robot's trunk frame O_b. The way these signals influence BlueFoot's gait control is covered in Chapter V. During navigation, the pitch of the trunk, $\theta_{b,x}$ is also controlled via its respective reference signal, $\theta^r_{b,x}$, via an outer-loop proportional control scheme. Control over trunk pitch allows for additional articulation of the LIDAR and camera sensors mounted to BlueFoot's head (trunk) while tracking features or composing 3D point clouds from LIDAR scans, as will be described later in this chapter.

Namely, BlueFoot's primary navigation mechanism fuses navigation reference commands generated using both LIDAR data and features from processed camera images. This algorithm is used as a *wandering* mechanism which would normally be employed as a first-level measure during situations where the robot has no information about its environment (*i.e.*, environmental map data, or landmark locations) *a priori*. This section will first describe the potential-fields portion of BlueFoot's base-navigation algorithm, which utilizes only LIDAR sensor data. The efficacy of this navigation mechanism will be supported with results from simulated trials. The incorporation of processed camera data will then be detailed to complete the description of the full navigation controller. A set of associated results will be presented from real-world trials.

6.1.1 LIDAR-based Potential-Fields Algorithm

The potential-fields portion of BlueFoot's navigation routines takes planar LIDAR scans as an input and generates a set of navigation outputs, $\vec{u}_L^r \in \Re^3$, which represents a direction of travel relative to the world frame, O_0, and P_L, which represents a total positive (attractive) potential. Each point from a LIDAR scan is mapped to a corresponding scalar *potential* which is used to influence the direction of the newly generated command. Given a LIDAR scan, S^L, with 2D scan points, $x_i^L \in S^L$, relative to the LIDAR's local coordinate frame O_L, first-level command elements are generated using a potential function $\{f(x) : \Re^3 \rightarrow \Re^1\}$, and a biasing function, $\{g(x, \psi) : \Re^3 \rightarrow \Re^1\}$, as follows:

$$\begin{aligned} \vec{u}_L^r &= \sum_{x_i^L \in S} g(x_i^L, \psi) f(x_i^L) \frac{x_i^L}{\|x_i^L\|} \\ P_L &= \alpha_p \sum_{x_i^L \in S^L} g(x_i^L, \psi) f(x_i^L) U\left(f(x_i^L)\right) \end{aligned} \tag{6.1}$$

The piecewise potential function, defined in (6.2), is used in BlueFoot's navigation scheme. This function is designed to repel the platform from objects which are closer to the robot than some minimum distance, d_{min}, and attract the robot toward objects that are further away. The form of this potential function was guided by several candidate force-field functions presented in [42]. The intention of (6.2) is to draw the robot towards long apertures, such as corridors or openings, and away from close-by obstructions. It is written as follows:

$$\begin{aligned} \Delta d &\equiv \|x\| - d_{min} \\ f(x) &= \begin{cases} -\lambda_{c,1} \left(\Delta d\right)^2 & \text{if } \Delta d < 0 \\ (\Delta d)\left(1 - e^{-\lambda_{c,2}(\Delta d)^2}\right) & \text{else} \end{cases} \end{aligned} \tag{6.2}$$

where $\lambda_{c,1} > 0$ and $\lambda_{c,2} > 0$ are tuning parameters used to specify the output range and sensitivity of the potential function output with respect to Δd, respectively. It can be observed that this potential function exhibits $f(x) < 0$ when $\|x\| < d_{min}$ and vice-verse, thus achieves the desired attractive and repulsive characteristics. Namely, this potential function favors points which are generally much further away from the robot, and applies *strong* repulsive forces only when obstacles come within a close range with the platform. These characteristics offer a higher propensity for exploration when the area being navigated is very spacious (with few obstacles in view), while encouraging *tight*

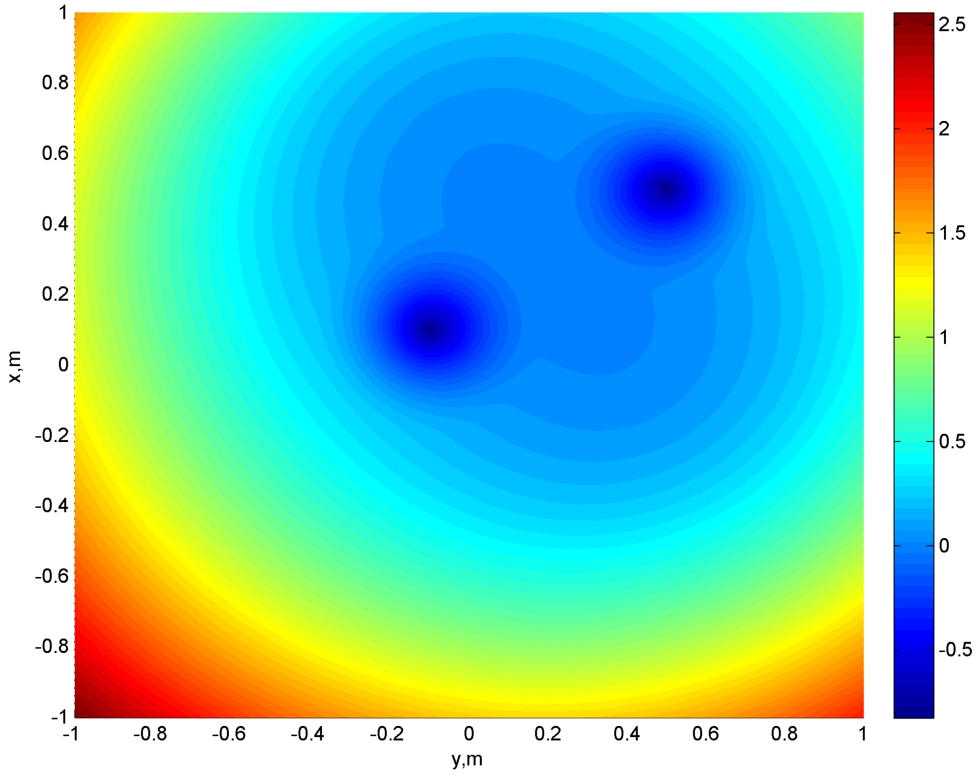


Figure 33: Potential field, $f(x)$ with $d_{min} = 0.4m$ given an environment with with obstacles.

navigation around from potential obstacles. A visualization of the potential field, $f(x)$, generated around two cylindrical obstacles (located at $(0.5, 0.1)$ m and $(0.1, -0.1)$ m from the robot) is shown in Figure 33 for a 2-by-2 meter area about the robot. It is notable that there exists largely negative potential values around the neighborhood of each obstacle, which result in repulsive force contributions.

The biasing function $g(x, \psi)$ is used to weight the effect of individual points from a single scan on the final output, with respect to their relative angular position in O_L. The biasing function is parametrized with an angular-window parameter $\psi > 0$. The function is used in BlueFoot's navigation function, which is defined as follows:

$$\begin{aligned} \mathrm{ang}\,(x) &= \tan^{-1}\left(\frac{[x]_x}{[x]_y}\right) \\ g(x,\psi) &= \begin{cases} \left(1 - \left\|\frac{\mathrm{ang}(x)}{\psi}\right\|\right)^{\alpha} & \mathrm{ang}\,(x) < \psi \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{6.3}$$

where $[x]_x$ and $[x]_y$ are the x-axis and y-axis coordinates of the vector argument $x \in \Re^3$; and $\alpha \leq 1$ such that $g(x,\psi)$ is concave with respect to α. This function is used to give priority points which exists in a frontward facing angular window, spanning 2ψ, centered at the y-axis which emanates from O_L.

Finally, outputs of the potential field algorithm are mapped into forward and turning rate commands, v_L^r and ω_L^r, respectively, by the following proportional control scheme:

$$\begin{aligned} \theta_L^r &= \tan^{-1}\left(\frac{[\vec{u}_L^r]_x}{[\vec{u}_L^r]_y}\right) \\ \dot{v}_L^r &= \beta_v\left(P_L + v_{L,min}^r - v_L^r\right) \\ \dot{\omega}_L^r &= \beta_\omega\left(\left(\frac{\omega_L^{r,max}}{\pi}\right)(\theta_L^r - \theta_{b,z}) - \omega_L^r\right) \end{aligned} \tag{6.4}$$

where β_v and β_ω are proportional-gain tuning parameters; $v_{L,min}^r > 0$ is a small, minimum velocity used to prevent the robot from getting stuck in local minima; and $\theta_{b,z}$ is robot's yaw in O_0. The parameters β_v and β_ω can be viewed as *update-inertias*, as they directly effect the influence of instantaneous commands on the forward velocity and turning rate of the robot. Furthermore, these gains can be used to low-pass updates to navigation parameters so as to remove jitter caused by degenerate LIDAR scans.

6.1.2 Simulated potential field navigation results

Figure 34 shows the path resulting from a simulated trial in which the BlueFoot robot autonomously wandered through an environment, consisting of a room and several long corridors. During this simulation, BlueFoot is navigated using the previously described potential-fields mechanism. A plot of the robot's trajectory (shown in yellow) shows that the robot was able to successfully navigate its immediate environment while avoiding collisions with the surrounding walls. Notably, the robot naturally avoided smaller out-cove regions (which were mostly filled with obstacles) because these regions had low relative potential which generated repulsive forces. A second set of results is

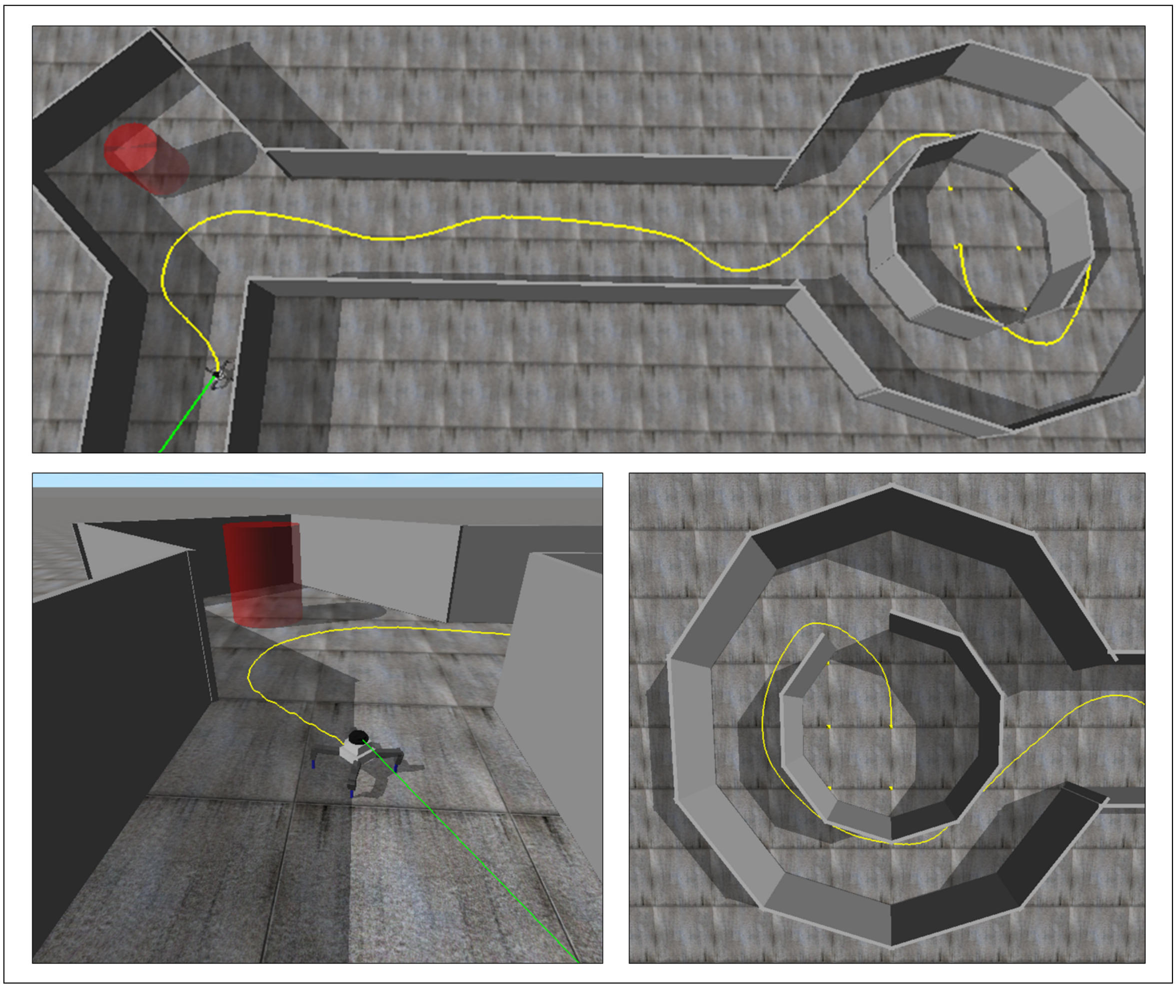


Figure 34: Path (shown in yellow) taken by robot through a simulated set of rooms and halls using the LIDAR-based potential fields navigation scheme.

shown in Figure 35 shows the robot's performance in environments additional obstacles. Likewise, the robot exhibits the desired obstacle-aversion performance using the describe potential field function.

6.1.3 Incorporation of Camera-based Feature-Tracking

Camera-based goal tracking is used in conjunction with the aforementioned potential fields navigation scheme to move the platform through an environment while seeking or tracking a particular target. In this case, targets take the the form of features extracted from processed camera data. The robot is guided towards these features using a simple visual-serving approach.

Trackable camera features can be generated in a variety of ways. For the purpose

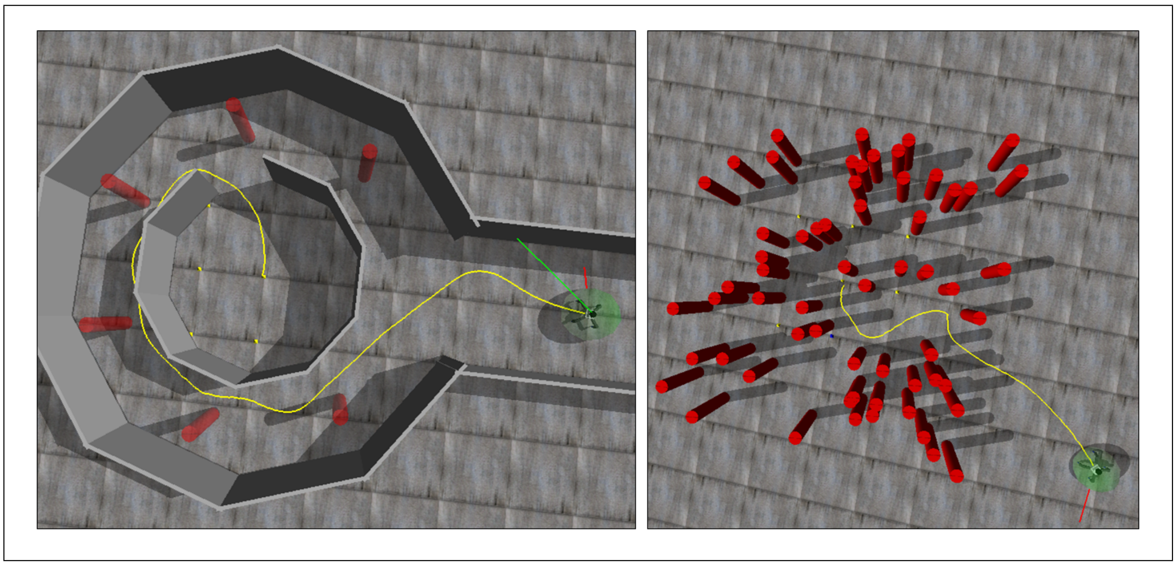


Figure 35: Path (shown in yellow) taken by robot through a simulated set of rooms and obstacles.

of BlueFoot's navigation, objects with distinct shapes or color have been chosen for tracking so that shape and blob detection algorithms easily can be employed to detect the relative positions of these features in BlueFoot's viewing range. Namely, Hough Transform-based shape detection and standard color-blob detection algorithms from the Open Computer Vision (OpenCV) library are employed for the purpose of feature detection [48]. Once detected, points representing the centers of each trackable feature, relative to the 2D camera-viewing frame, are mapped into forward velocity and turning rate commands. These command are used to control the robot such that the center of the feature is aligned with the center of each image fame. These camera-based commands are then mixed with the outputs of the potential-fields controller as a weighted sum to form a hybrid navigation control law.

The visual-servoing approach to be described is agnostic of the type of feature being tracked. Namely, this approach relies on the relative position of the center of each feature, represented as a pixel-position, $p_{Im} = [u, v]^T \in \mathbb{Z}^2$ in the 2D image frame, O_{Im}, and a relative size, r, measured in pixels. In the case of circular features, for example, r represents the radius of the detected circle. For color-blob features, r represents the radius of a circle which fully inscribes the colored object.

For the purpose of target tracking, it is desired that the robot's forward speed be controlled such that is proportional to r. Namely, it is desired that the robot stops when it becomes *close enough* to the target object, and faster towards the goal when the feature is in sight but the robot is further away. The position of the feature's center is used to control the robot's turning rate, as well as the commanded pitch of the robot's trunk, $\theta^r_{b,x}$. Trunk articulation important during feature tracking routing as it aids in keeping the tracked-target objects centered in the image frame. Provided that the target is moving slower than the what the system can track, this will ensure the target remains in sight at all times.

A separate set of navigation commands, v^r_C and ω^r_C; and an additional body-pitching command, $\theta^r_{b,x}$, are generated from an extracted feature location, $p_{Im} = [u, v]^T$ as follows:

$$\begin{aligned} v^r_C &= v^{r,max}_C \left(1 - e^{-c_r(r - r_{min})^2}\right) \\ \omega^r_C &= \omega^{r,max}_C \left(\frac{w_{Im} - 2u}{w_{Im}}\right) \\ \theta^r_{b,x} &= \theta^{r,max}_{b,x} \left(\frac{2v - h_{Im}}{h_{Im}}\right) \end{aligned} \tag{6.5}$$

where $v^{r,max}_C$, $\omega^{r,max}_C$ and $\theta^{r,max}_{b,x}$ are the maximum magnitudes of forward velocity, turning rate, and body-pitching commands, respectively; c_r is a sensitivity parameter; r_{min} defines a minimum feature size which will result in the administration of a zero velocity command to the platform (and thus the distance from the feature at which to halt forward motion); and w_{Im} and h_{Im} define the width and height, respectively, of the image being processed. Having now established a formulation for how a single, distinct, feature is used to guide the platform towards a target, a means of fusing the LIDAR-based command signals and the camera-based command signals will be defined.

The composition of this hybrid command technique is motivated by two related subtasks:

1. to use the potential fields algorithm during a wandering phase, when a target object is not in sight and

2. to guide the robot safely towards the goal once in sight.

This causes camera-based tracking commands to have a greater influence on system navigation (through the variables v^r and ω^r) as the platform becomes closer to the desired target. A straight-forward way to achieve this is to use the relative size. r, of tracked target features in the image frame. With this in mind, a simple command

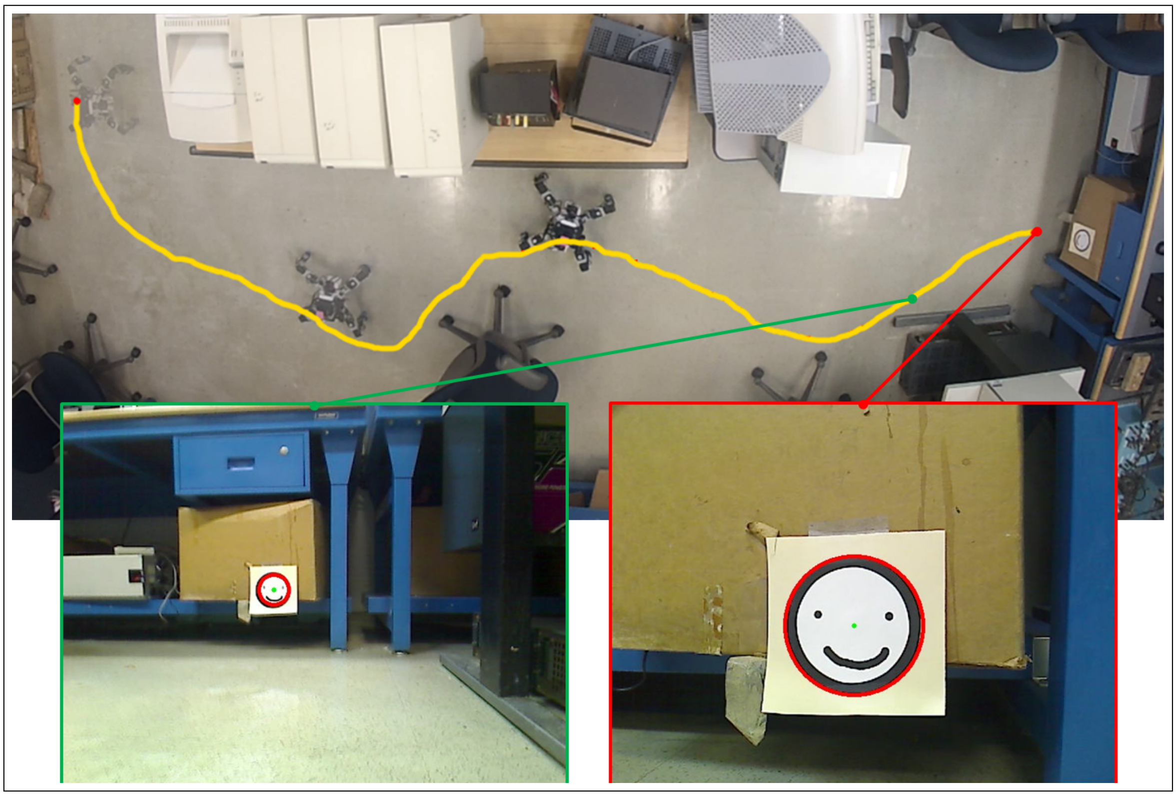


Figure 36: Path (shown in yellow) taken by robot through a real-life lab setting showing the robot's view of the goal marker.

mixture scheme has been defined as follows:

$$v(r) \equiv \begin{cases} e^{-c_{mix}(r-r_{mix})^2} & \text{if } r < r_{mix} \\ 1 & \text{else} \end{cases}$$

$$\begin{bmatrix} v^r \\ \omega^r \end{bmatrix} = v(r) \begin{bmatrix} v^r_C \\ \omega^r_C \end{bmatrix} + (1 - v(r)) \begin{bmatrix} v^r_L \\ \omega^r_L \end{bmatrix} \tag{6.6}$$

where c_{mix} is a sensitivity parameter; and r_{mix} defines the processed-feature size which will cause the robot to be navigated, mostly, using camera features. The reasoning for such a scheme involves a heuristic approach to obstacle aversion, which assumes that when the platform is further away from a goal, there is a higher probability that it will encounter an obstacle. Conversely, when the goal becomes closer to the platform, it is assumed that the number of obstacles between the robot and the target is lower, making it safe to shift full priority to reaching the goal from the current position. It is defined that if a feature is not visible within the current view of the camera, then $r = 0$.

Figure 36 shows the robot's path through a real-life environment. The efficacy of this navigation scheme is highlight, particularly, when the robot successful rounds the chair obstacle in its way. Additionally, the robot successfully detects and reaches a target marker during this exemplary trial. The robot's view of the target marker from further away, and when reached, are also shown in this figure. The robot highlights the perimeter of the marker in red and its center in green upon processing.

6.2 Towards Rough Terrain Navigation

6.2.1 Terrain Mapping from 2D scans

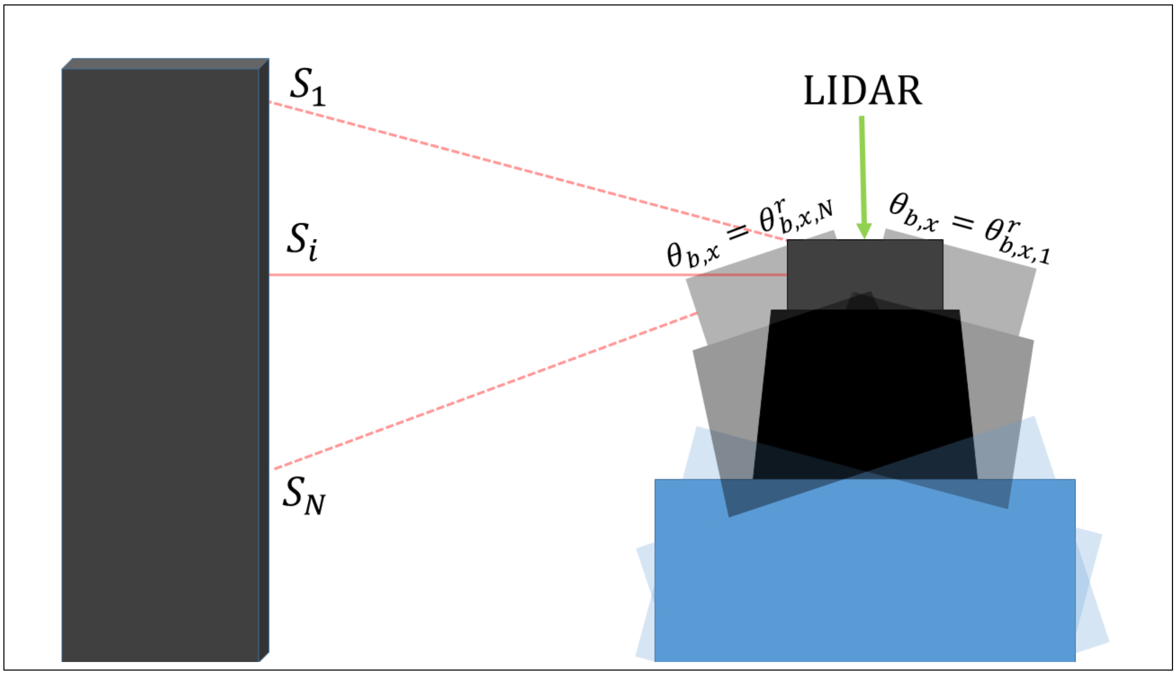


Figure 37: LIDAR swept over a range of angles, $\theta_{b,x} \in [\theta^r_{b,x,1}, \theta^r_{b,x,N}]$.

The BlueFoot platform has the ability to compose 3D point clouds from a series of swept 2D LIDAR scans in conjunction with a trunk orientation estimates, $\hat{\theta}_b$. LIDAR articulation is achieved by slowly pitching the trunk over some angular range while keeping the platform's feet ridgedly planted. Sweeping range is limited by the kinematic-feasibility of each trunk pose that must be reached during a sweep. Given a particular set of foot and body location, kinematic feasibility is validated using the inverse kinematics solution described in Section 4.1.2. A single 2D scan is taken at each pose within the body-sweep trajectory. The newly acquired scan is transformed from the LIDAR sensor frame, O_L, to the world frame by a homogeneous transformation H^L_0 which is defined as follows

$$H_0^L = H_b^L H_0^b \tag{6.7}$$

where H_0^b is a transformation from O_0 to the trunk frame O_b, as defined in Chapter IV; and H_b^L defines a transformation from the frame O_b to the LIDAR frame, H_b^L. H_b^L is necessary for knowing the position of the LIDAR head with respect world frame, as the sensor itself has some offset and rotation relative to the robot's body. Each 2D point from $x_i \in S$ from the initial scan, $S \subset \Re^2$, can then be transformed into a 3D scan segment, $\bar{S}_j$, in O_0 by:

$$\begin{bmatrix} \bar{x}_{i,j} \\ 1 \end{bmatrix} = H_b^L H_0^b \begin{bmatrix} x_i \\ 0 \\ 1 \end{bmatrix} \forall x_i \in S \tag{6.8}$$

where $\bar{x}_{i,j} \in \bar{S}_j$ is a point withing the 3D j^{th} scan segment $\bar{S}_j \subset \Re^3$. After the sweeping routine is complete, 3D scan segments are composed into a final point cloud, $\bar{S}$ by:

$$\bar{S} = \bigcup_{j=1}^{N_s} \bar{S}_j \tag{6.9}$$

where N_s defines the number of scans taken during the sweeping routine. For the sake of simplicity, it is assumed that the trunk's position, p_b, is fixed (system is completely ridged) during a swept-scan routine. In the results to be presented, this seems to be a reasonable assumption given that the platform is at rest and the trunk is pitched sufficiently slowly over the angular sweeping range. A slow sweep rate ensures that perturbations caused by vibrations incurred by trunk rotation and foot-slip are small, and thus do not cause for significant deviations in LIDAR scan points.

6.2.2 Height-map Generation from 3D point cloud

3D point clouds can be represented as height-maps in a fictitious height-map coordinate system, $O_{\mathcal{M}}$. These representations are convenient for use in planning as they can be used to assign costs to particular robot configurations during terrain traversal straightforward way. In implementation, height-maps are represented as a matrix $\mathcal{M} \in \Re^{n \times m}$ with height elements $m_{i,j}$. A set of indices $\bar{p} = \{i, j\}$ represents a discrete, planar location within the $O_{\mathcal{M}}$ frame. In the sections to follow, the notations $m_{i,j}$ and $\mathcal{M}(\bar{p})$ will be used interchangeably to represent a height value.

To create the height-map, a $w \times d$ region of interest (ROI) is discretized in (nm) subregions. The notation $\mathbb{R}^{\mathcal{M}} \subset \Re^3$ will be used to represent the set of points contained

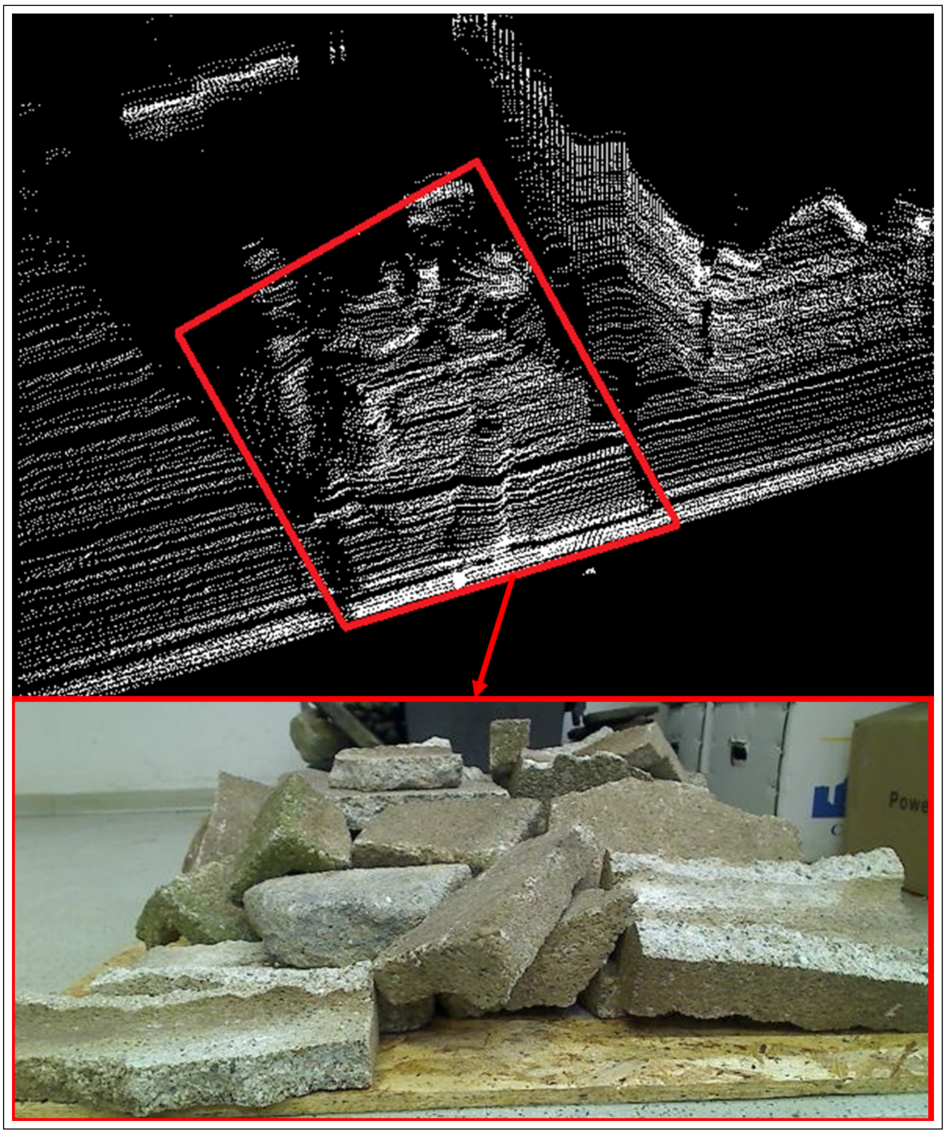


Figure 38: Original 3D point-cloud of terrain patch *(top)* and corresponding view from robot's on-board camera *(bottom)*.

within the ROI, and $\mathbb{Z}^{\mathcal{M}}$ will be used to represent its discretized analogue. An ROI from the source point cloud, $\bar{S}$, will be represent as $\bar{S}_{ROI} \subset \Re^3$. The location and size of this ROI would have to be determined with some auxiliary detection process. Such an algorithm must accomplish the following subtasks:

1. select an area with appropriately high terrain variation (large changes in gradient)

2. and determine the bounding region where this patch of rough-terrain exists

For the results shown in this section, the evaluated ROI has been manually selected after an inspection of an existing 3D point cloud, as an adequate algorithm has yet to be implemented which fulfills the requirements of the detection task described. Such an algorithm would need to involve mechanisms for point cloud segmentation and feature evaluation. Additionally, images of the the robot's immediate terrain could be used to aid in this process by locating areas with high feature density incurred by terrain variations.

The point $\bar{p} \in \mathbb{Z}^{\mathcal{M}}$ is used to represent the location of a discrete subregion that is $(w/m) \times (d/n)$ area. A corresponding height for that subregion is stored in $m_{i,j}$, and is selected as the largest z-component of any point that falls within said subregion. The transformation from a point cloud element, $\bar{x}$, to an element within the discretized height-map frame can be described by:

$$\begin{bmatrix} \bar{p} \\ m_{i,j} \end{bmatrix} = \begin{bmatrix} i \\ j \\ \hline m_{i,j} \end{bmatrix} = \left\lceil \begin{bmatrix} n/d & 0 & 0 \\ 0 & m/w & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \bar{x} + \begin{bmatrix} 1.5d \\ 0.5w \\ 0 \end{bmatrix} \right) \right\rceil \tag{6.10}$$

where $\lceil * \rceil$ is an element-wise vector ceiling function.

Depending on the density of the source point cloud, the aforementioned conversion process can produce relatively sparse height-maps. To deal with this, a dilation and smoothing routine is used to fill in gaps in $\mathcal{M}$. During dilation, each non-zero height element within the map is expanded into a region around an existing element with non-zero value [65]. Dilation parameters are tuned so that semi-uniform map with minimal gaps between non-zero height elements is achieved. Finally, a median filter is applied to the dilated height-map to smooth transitions between height elements. The smoothing operation performed by the median filter is performed by replacing all values within a window of elements (usually forming a square region) with the middle-valued element within said window. The full 3D point cloud to height-map conversion is summarized in Algorithm 2. In this algorithm, *isolateROI* represents a function which locates a region of interest within the point cloud $\bar{S}$; *subdivIndex* performs the transformation in (6.10), which generates indices within the frame $O_{\mathcal{M}}$ from points in O_0; *dialateFeatures* performs a tuned image-dilation routine; and *medianFilter* performs a standard image median filter routine with a $w \times w$ window size. Once a height-map has been generated, a corresponding gradient, as shown in Figure 39, is generated using a Sobel image gradient operation (from OpenCV).

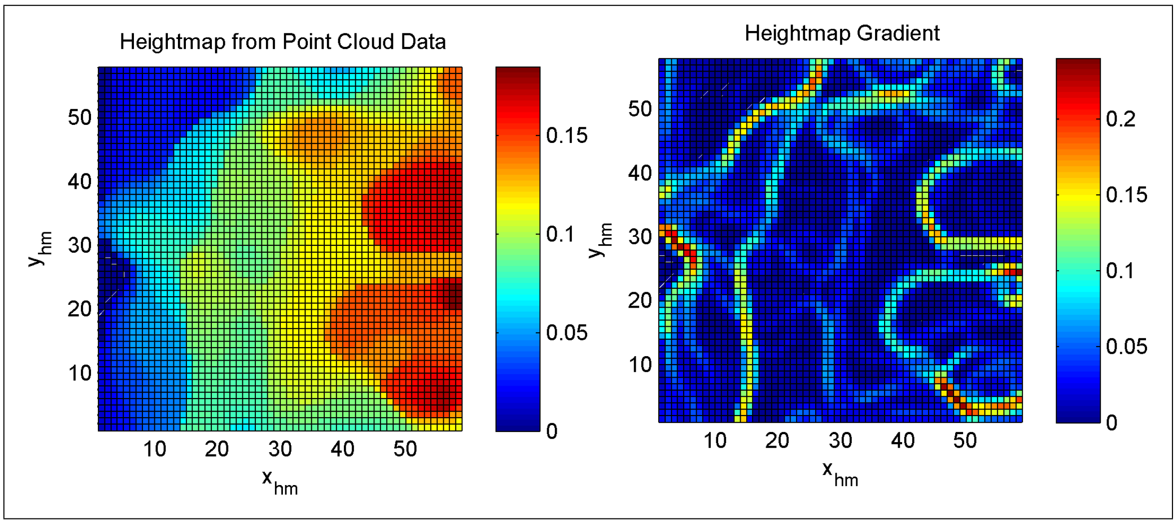


Figure 39: Relative height-map *(left)* and its corresponding gradient *(right)*.

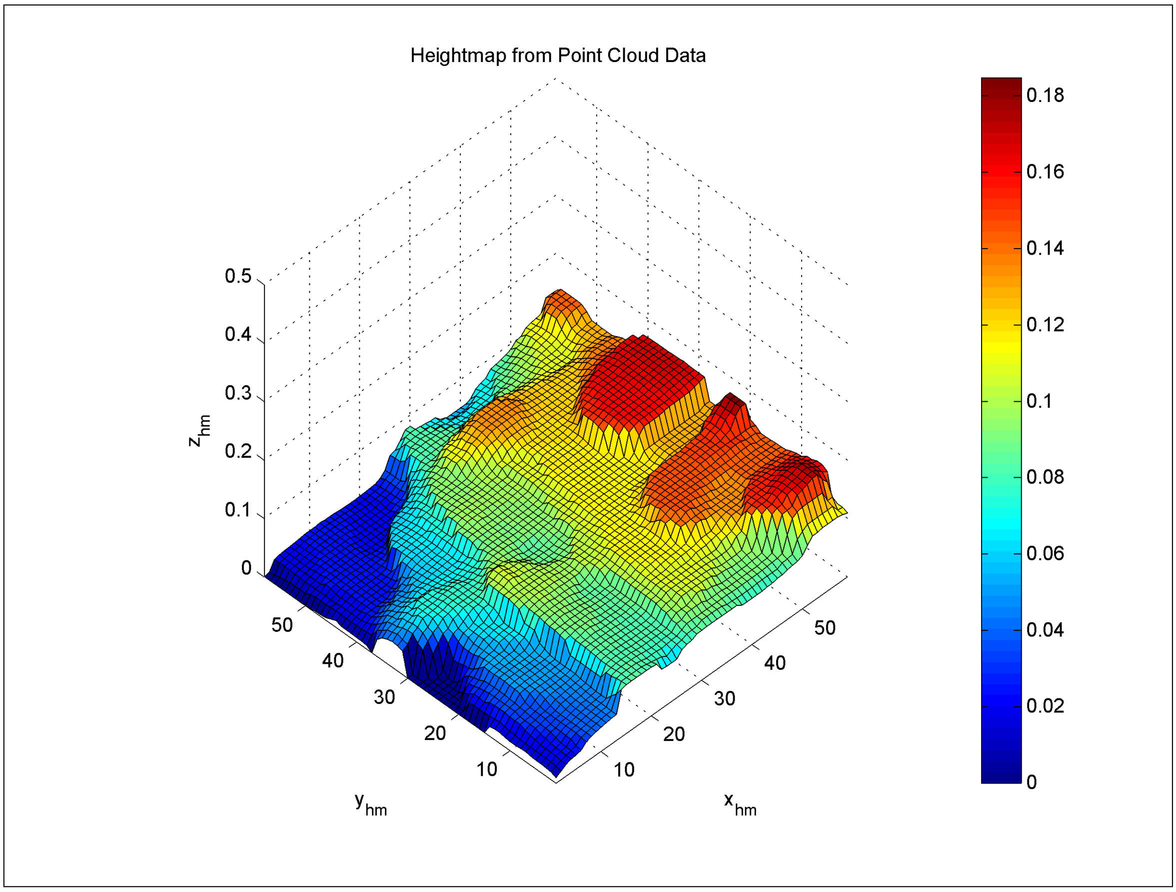


Figure 40: Height-map showing terrain variation in the $z_{\mathcal{M}}$ direction.

6.2.3 Generating a Cost-map from the Height-map

A discrete height-map space can be converted to a cost-space representation for the purpose of foot-placement planning over rough terrain. Here, a discrete cost-map

Algorithm 2 3D ROI point cloud to height-map conversion.

```
init $\mathcal{M} = \Theta, w, d, \bar{S}$
$\bar{S}_{ROI}$ = isolateROI($\bar{S}$)
for all $\bar{x}_k \in \bar{S}_{ROI}$ do
    $[i, j]$ = subdivIndex($[\bar{x}_k]_x, [\bar{x}_k]_y, w, d$)
    if $[\bar{x}_k]_z > m_{i,j}$ then
        $m_{i,j} \leftarrow [\bar{x}_k]_z$
    end if
end for
$\mathcal{M} \leftarrow$ dialateFeatures($\mathcal{M}$)
$\mathcal{M} \leftarrow$ medianFilter($\mathcal{M}, w$)
```

is represent by $\mathcal{C} \in \Re^{n\times m\times l}$. The first and second dimensions of $\mathcal{C}$ (of size n and m) directly correspond to the width and depth of the discretized height-map space $\mathbb{Z}^{\mathcal{M}}$. The third dimension represents a discretization of the robot's yaw state, $\theta_{b,z} \in [-\pi, \pi]$ into l subdivisions, and is used to represent variations of the robot's yaw with respect to $O_{\mathcal{M}}$ in the height-map space. We define a mapping from the height-map space to a the cost-space as follows:

$$\left\{\mathcal{C} = f(\mathcal{M}, \Gamma) \ : \ \Re^{n\times m} \rightarrow \Re^{n\times m\times l}\right\} \tag{6.11}$$

where Γ is a static configuration matrix used to describe the nominal spacing between the robot's feet. Γ is composed as follows:

$$\Gamma \equiv \left\{\bar{p}^{\mathcal{M}}_{1,e}, \bar{p}^{\mathcal{M}}_{2,e}, \bar{p}^{\mathcal{M}}_{3,e}, \bar{p}^{\mathcal{M}}_{4,e}\right\}$$

where $\bar{p}^{\mathcal{M}}_{v,e} = [i_v, j_v]^T \in \mathbb{Z}^{\mathcal{M}}$ represent the position of each v^{th} foot in the height-map space with $i_v \in \{1, ..., m\}$ and $j_v \in \{1, ..., n\}$. By formulating cost with respect to static foot configurations, planning is simplified to finding a trajectory for body position. Target leg positions are then generated from planned body locations by Γ.

This conversion mapping, $f(\mathcal{M}, \Gamma)$, has been formulated with respect to three main cost elements:

1. variation in attainable foothold heights between all feet

2. terrain steepness around any foothold, given a particular configuration

3. and net robot rotation performed at each path-step.

The first cost element is chosen to simplify to problem of ensuring the robot plans to walk over relatively level terrain by penalizing height variation between planned foothold location. This also aids in ensuring that the kinematic workspace of each leg is not compromised by reducing the amount of "stretching" the robot has to do to attain particular configuration upon the terrain. The second cost element is used deter the robot from attempting to travel atop overly steep terrain. The final cost element is used to ensure the robot does not plan to perform any large changes in direction while traversing the terrain, which could cause for needless motion (perhaps even loops) in the planned path.

To define perform the conversion between $\mathcal{M}$ and $\mathcal{C}$, we first define a *moving* foothold matrix $\Gamma'(\bar{p}_b^{\mathcal{M}}, u) \in \mathbb{Z}^{2\times 4}$, as follows:

$$\Gamma'(\bar{p}_b^{\mathcal{M}}, u) = \left\lceil R_z^{\mathcal{M}}(\gamma(u))\Gamma \right\rceil + \bar{p}_b^{\mathcal{M}} B \tag{6.12}$$

where $\bar{p}_b^{\mathcal{M}} = [i, j]^T \in \mathbb{Z}^{\mathcal{M}}$ and $u \in \{1, ..., l\}$ are used as a discrete representation of the robot's trunk position and yaw within the cost-space, respectively; $\{\gamma(u) : \mathbb{Z} \to \Re\}$ represents a linear mapping from the index u to a robot yaw in $O_{\mathcal{M}}$; $R_z^{\mathcal{M}}(\gamma(u)) \in \Re^{2\times 2}$ represents a rotation matrix about the z-axis in $O_{\mathcal{M}}$; and $B = [1, 1, 1, 1]$.

The cost-map is then generated as follows:

$$\begin{aligned}
\Gamma' &\equiv \Gamma'(\bar{p}_b^{\mathcal{M}}, u) \\
\mathcal{H}_v &= \mathcal{M}(\mathrm{col}(\Gamma')_v) \\
\delta\mathcal{H}_v &= \nabla\mathcal{M}(\mathrm{col}(\Gamma')_v) \\
\mathcal{C}(\bar{p}_b^{\mathcal{M}}, u) &= k_{\mathrm{var}}\mathrm{var}(\mathcal{H}) + k_\delta \sum_{v=1}^{4} \delta\mathcal{H}_v^2 + k_\theta \gamma^2(u)
\end{aligned} \tag{6.13}$$

$\forall \bar{p}_b^{\mathcal{M}} \in \mathbb{Z}^{\mathcal{M}}$, $\forall u \in \{1, ..., l\}$, and $v \in \{1, 2, 3, 4\}$, where $\mathcal{H}$ is a collection heights at each foothold position in $\mathcal{M}$; $\delta\mathcal{H}$ is a collection of corresponding elements from the gradient matrix $\nabla\mathcal{M}$; var$(*)$ is generates the variance between the elements of a vector argument $(*)$; k_{var}, k_δ and k_θ are scalar weighting parameters; and col$(*)_v$ extracts the v^{th} column from the matrix argument $(*)$. Figure 41 shows sample cost-map visualizations for fixed robot orientations of $\theta_z = 0$ and $\theta_z = \pi/4$. This cost map was generated using the height-map and corresponding height-map gradient shown in Figure 39.

$\mathcal{C}$ is used to generate an optimal, N-step path, which is comprised of N, discrete robot configurations. Moreover path is planned over rough terrain by optimizing a finite,

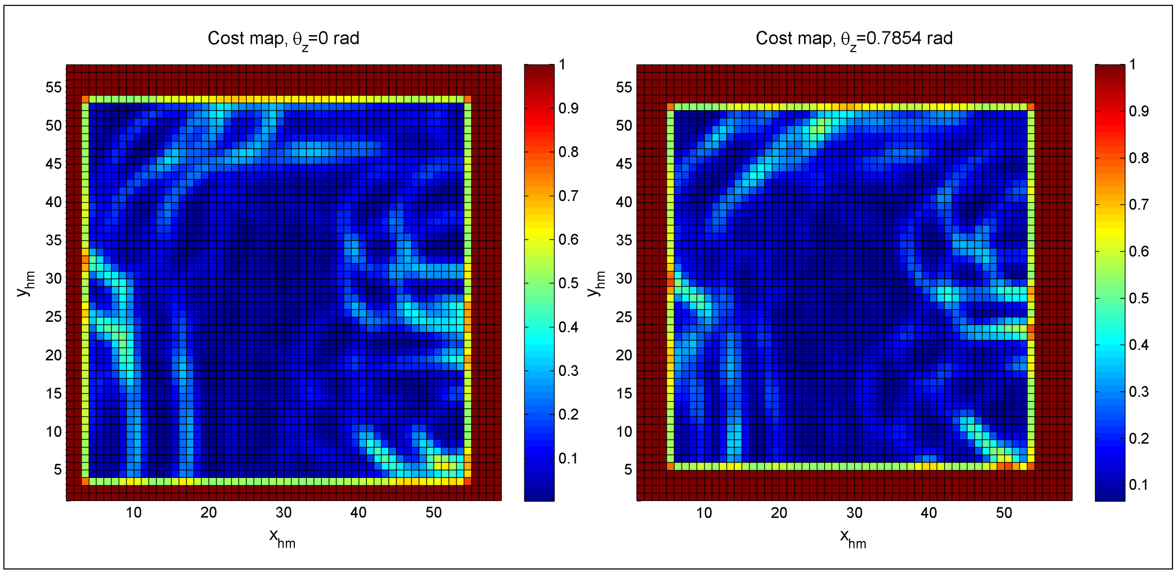


Figure 41: Projections of a sample cost map generated from the height-map shown in Figure 39.

N-step cost functional defined as follows:

$$J(N) = \sum_{k=0}^{N} \mathcal{C}\left(\bar{p}^{\mathcal{M}}_{b,k}, u_k\right) \tag{6.14}$$

An optimal path can be found over the discretized space using an A^*-based path planning algorithm, for example. Additionally, sub-optimal paths could be generated using a potential fields approach. Both of these directions have yet to be fully explored as part of this project. Paths generated from the cost map would be mapped back into O_0, using the corresponding inverse mappings of (6.10) and $\gamma\,(u)$, and interpolated to generate smooth trajectories.

6.2.4 Surface Reconstruction

3D surface reconstruction is carried out according to a process for normal-estimation from 3D point cloud data detailed in [44]. According to this procedure, 3D surfaces are reconstructed from point cloud data, $\bar{S}$, by fitting a collection of flat polygonal elements to the source point cloud. Each of these estimated elements is represented by a surface-normal which emanates from a corresponding point within the source cloud. Surface normal estimation carried out by fitting planes to groups of points which exist within a neighborhood d_s about each point $\bar{x}_i \in \bar{S}$.

Before normal estimation, the point cloud $\bar{S}$ (which is usually dense) is down-sampled and smoothed before a normal estimation algorithm is applied. Point cloud down-sampling is performed using a voxel-grid technique, which discretizes the point cloud space into a voxel-space. The voxel-space consists of cuboids of a particular width, depth and height. Here, a cubic voxel space is used parameterized by a single voxel side-length dimension d_{vox}. The voxel grid filter and generates a single point from all points which fall within the volume of a voxel region, typically as a mean of all such points. The location of points which occupy the voxel region is aided by a KD-tree search technique, which offers higher efficiency over a brute-force search for neighboring points. This voxel-grid down-sampling technique produces a reduced representation of the original cloud, $\hat{S}$, which has shown, though observation of empirical results, to contain up to ten to twenty times less points, $\bar{S}$. This reduced-point cloud representation serves to greatly reduce the computational burden of successive cloud processing operations.

Algorithm 3 Finding good places to step from a 3D point cloud.

init $\bar{S}, \bar{S}^*, d_{vox}, \vec{u}^r, e, e_{max}$
$\hat{S} = \text{voxelGridFilter}(\bar{S}, d_{vox})$
$\hat{S} \leftarrow \text{movingLeastSquaresFilter}(\hat{S})$
$\hat{N} = \text{estimateNormals}(\hat{S})$
for all $\vec{n}_i \in N$ **do**
  $e = 1 - (\vec{u}^r)^T \vec{n}_i$
  if $e < e_{max}$ **then**
    $\bar{S}^* \leftarrow \bar{S}^* \cup \bar{x}_i$
  end if
end for

Next, the resulting cloud, $\hat{S}$, is regularized using a moving least-squares filter. This filter creates a smoothed point cloud by performing a sequence polynomial fits over a moving subset of points. As previously mentioned, normals are estimated from the smoothed point cloud via a plane fitting method, as described in [66]. Planes are estimated using Principle Component Analysis (PCA) on clusters of points [67]. The standard PCA algorithm provides a least-squares plane fit by way of dimensional reduction, yielding best-fit description for a set 3D points by way of a 2D manifold [46]. The full surface-normal estimation algorithm is provided in Algorithm 1. A point cloud with estimated surface normals is shown in Figure 42.

Algorithm 3 is used convert a 3D point cloud into a point-cloud with normal vectors, which is then used to find "flat" regions within the reconstructed surface. This is performed for the purpose of footstep planning. Flat regions are located using generated surface, whose alignment is graded against a reference unit vector $\vec{u}^r$ by taking the dot product between $\vec{u}^r$ and a surface normal. This dot-product operation produces an alignment error $e \in [0, 2]$. Points with associated normals whose alignment error is less than a scalar error bound, e_{max}, are selected as fit surfaces for walking. Obviously, not all surfaces which satisfy this alignment criteria represent surfaces which are fit for traversal. Beyond normal estimation, several post processing operations will be performed to segment the resulting cloud into near-by surfaces over which the robot can walk.

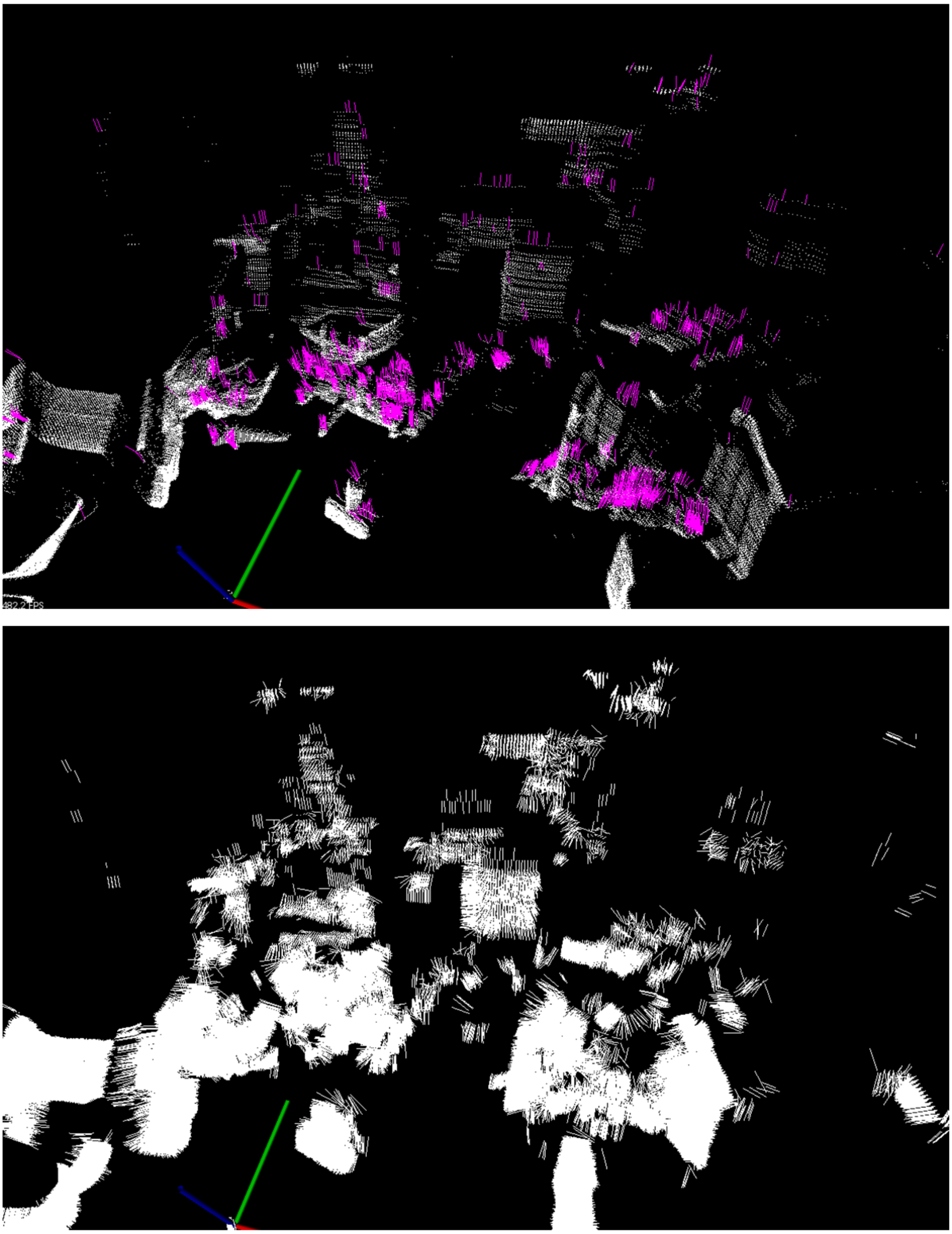


Figure 42: Point cloud generated from sequential scans of a room showing flat-surface candidates *(top)* and 3D Point Cloud with normal estimation *(bottom)*.

CHAPTER VII

Concluding Remarks

This thesis has provided a comprehensive summary of the design and control of the BlueFoot quadruped platform. Namely, BlueFoot's structural makeup; component devices; software architecture; gait and stability control; and related navigation control mechanisms have been described in detail. Results from simulated and actual trials have been provided to demonstrate the performance of routines used to control the BlueFoot platform. This final chapter will provide some insight about future directions for the BlueFoot project, including future directions with work regarding the design of the BlueFoot platform; and with work regarding its control and navigation.

Future work regarding the design of the existing BlueFoot platform will focus on redesigning the structure of BlueFoot's legs. Namely, the platform currently requires servos which generate higher torque outputs, particularly at the upper joints. The torque output of BlueFoots's current joint servos has shown to be adequate for moderate locomotion tasks, but may not be powerful enough for more demanding tasks over extended periods of operation. Currently, servos which actuate the upper joints tend to overheat during operation, causing them to shut down. Joint shut down is a saftey feature which is built-in to the smart servo control software. This indicates that they are constantly operating near the output limits during moderately demanding tasks. It would also be beneficial to re-incorporate series elastic joints into the robot's design for the purpose of generating more *organic* gait motions and relieving dynamic torque loading during impact. The incorporation of series elastic joints would also allow for faster gaits and extended this project toward possible future studies regarding under-actuated quadruped gait control.

Future work involving the control of the BlueFoot platform will mainly focus on research into algorithms for optimal motion planning and traversal over irregular terrain, as well as optimal whole-body motion planning towards sensor articulation and dynamic gait stability applications. In regards to rough-terrain planning, supporting algorithms must also be developed for the classification of rough terrain regions. Such a routine will be necessary for deducing whether or not an area of terrain is of high irregularity; as well as whether or not the terrain is traversable or needs to be traversed in order to reach a goal location. The associated area will be mapped and the surface reconstruction mechanisms details in this thesis will be employed to represent the region as a collection of terrain features. Finally, an adaptive footstep and body-placement planner will be used to move the robot over rough terrain.

The ultimate goal of this research is to design a set of control-laws for the BlueFoot platform such that is can simultaneously and incrementally map and update its foot placement and planned-path for navigation in real time. There are many situations wherein incremental terrain mapping and re-planning is necessary, especially in situations were information about upcoming terrain is incomplete. Specifically, one such situation occurs when the robot can only partially perceive upcoming terrain from its current vantage point. This could occur if the terrain being mapped was taller than the effective height of the LIDAR laser head, and the sensor could not be articulated in such a way that the entirety of an immediate terrain patch was viewable.

In the event of missing terrain data, current implementations utilize approximations the for terrain beyond the portions which are immediately viewable by robot's vision sensors, such as that described in [68]. Instead of taking a direction towards improving these types of techniques, it would be more important develop a set of techniques which can be used to navigate the robot based on a continuously updated terrain map. In BlueFoot's case, this would specifically mean that the robot must execute a coordinated set of motions which allow it to articulate its LIDAR sensor, for the purpose of effective terrain mapping, while also walking over said terrain. From this, a complicated whole-body motion planning problem arises. The solution to such a problem would need to address a method for generating optimal body-motion trajectories which simultaneously achieve a stable robot configuration while also generating effective sweeping motions for terrain scanning. This problem also demand accurate localization of the robot from terrain features, as vision sensors would be used to generate new knowledge about the walking surface, in particular. For this, methods for simultaneous localization and mapping in 3D will have to be explored. Additionally, terrain features could be used to localize the robot based on its kinematic configuration. In such a localization scheme, robot orientation, foot contact, and joint position data will be used to estimate its kinematic pose, which would then be used to find a possible set on the surface of a known terrain element which could represent current foot contact points. For this task, localization via a particle filtering scheme could be employed, in which multiple estimates of the robot upon a section of terrain are maintained.

Additional problems arise from the root rough-terrain planning problem, including an assignment of cost to the act of actually mapping a region of terrain. In this regard, one must consider the time and effort (energy demands, etc.) required for completing a terrain map of a particular region. In BlueFoot's case, a significant amount of general control *effort* is spent during terrain scanning. If we consider a routine similar to BlueFoot's current terrain-scanning routine, in which the robot comes to a complete stop before sweeping its body to generate varying scan levels, then the cost of generating information about the upcoming terrain is obvious in that it manifests in both the *time* and physical *energy* spent collecting terrain information. In a more elegant control scheme, where the robot simultaneously moves and maps its terrain, this cost manifests as extra energy exerted in the act of articulating the robot's body. From these considerations arises a question about how a robot entity can somehow predetermine the cost of mapping an area of terrain, from a smaller set of terrain features, before exerting the full, required effort acquiring knowledge about a particular region in its environment. The robot could also incorporate a determination about which terrains have a higher potential for successful traversal into this mapping-effort cost. Navigation control could then be achieved as a fusion between optimal robot body planning in concert with optimal deduction about the path to travel as a function of the effort necessary to overcome its associated terrain.

Appendix A

Leg Jacobian with respect to frame $O_{i,0}$

$$
\begin{aligned}
J_{i,e}^{i,0} &= \begin{bmatrix} j_{1,1}^{i,0} & \cdots & j_{1,4}^{i,0} \\ \vdots & \ddots & \vdots \\ j_{6,1}^{i,0} & \cdots & j_{6,4}^{i,0} \end{bmatrix} \\
j_{1,1}^{i,0} &= -s_{1,i}(2a_1 + 2a_3c_{23,i} + 2a_2c_{2,i} + a_4c_{234,i})/2 \\
j_{1,2}^{i,0} &= -c_{i,1}(a_3s_{23,i} + a_2s_{i,2} + a_4s_{234,i}/2) \\
j_{1,3}^{i,0} &= -c_{i,1}(a_3s_{23,i} + a_4s_{234,i}/2) \\
j_{1,4}^{i,0} &= -a_4s_{234,i}c_{i,1}/2 \\
j_{2,1}^{i,0} &= c_{i,1}(2a_1 + 2a_3c_{23,i} + 2a_2c_{2,i} + a_4c_{234,i})/2 \\
j_{2,2}^{i,0} &= -s_{1,i}(a_3s_{23,i} + a_2s_{i,2} + (a_4s_{234,i})/2) \\
j_{2,3}^{i,0} &= -s_{1,i}(a_3s_{23,i} + a_4s_{234,i}/2) \\
j_{2,4}^{i,0} &= -a_4s_{234,i}s_{1,i}/2 \\
j_{3,2}^{i,0} &= a_3c_{23,i} + a_2c_{2,i} + a_4c_{234,i}/2 \\
j_{3,3}^{i,0} &= a_3c_{23,i} + a_4c_{234,i}/2 \\
j_{3,4}^{i,0} &= a_4c_{234,i}/2 \\
j_{4,2}^{i,0} &= s_{1,i} \\
j_{4,3}^{i,0} &= s_{1,i} \\
j_{4,4}^{i,0} &= s_{1,i} \\
j_{5,2}^{i,0} &= -c_{i,1} \\
j_{5,3}^{i,0} &= -c_{i,1} \\
j_{5,4}^{i,0} &= -c_{i,1} \\
j_{6,1}^{i,0} &= 1 \\
j_{6,4}^{i,0} &= j_{6,3}^{i,0} = j_{6,2}^{i,0} = j_{4,1}^{i,0} = j_{5,1}^{i,0} = j_{3,1}^{i,0} = 0
\end{aligned}
$$

[1] R.B. McGhee and A.A. Frank. On the stability properties of quadruped creeping gaits. *Mathematical Biosciences*, 3(0):331 – 351, 1968.

[2] J.K. Hodgins and M. Raibert. Adjusting step length for rough terrain locomotion. *IEEE Transactions on Robotics and Automation*, 7(3):289–298, Jun 1991.

[3] R. Altendorfer, N. Moore, H. Komsuoglu, M. Buehler, Jr. Brown, H.B., D. McMordie, U. Saranli, R. Full, and D.E. Koditschek. RHex: A Biologically Inspired Hexapod Runner. *Autonomous Robots*, 11(3):207–213, 2001.

[4] J.Z. Kolter, M.P. Rodgers, and A.Y. Ng. A control architecture for quadruped locomotion over rough terrain. In *IEEE International Conference on Robotics and Automation*, pages 811–818, May 2008.

[5] Tedrake R. Kuindersma S. Wieber, P.-B. Modeling and control of legged robots. In Siciliano and Khatib, editors, *Springer Handbook of Robotics, 2nd Ed*, Lecture Notes in Control and Information Sciences, chapter 48. Springer Berlin Heidelberg, 2015.

[6] Y. Fukuoka, H. Kimura, Y. Hada, and K. Takase. Adaptive dynamic walking of a quadruped robot 'Tekken' on irregular terrain using a neural system model. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 2037–2042 vol.2, Sept 2003.

[7] Joaquin Estremera and Kenneth J. Waldron. Leg Thrust Control for Stabilization of Dynamic Gaits in a Quadruped Robot. In Teresa Zieliska and Cezary Zieliski, editors, *Romansy 16*, volume 487 of *CISM Courses and Lectures*, pages 213–220. Springer Vienna, 2006.

[8] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. BigDog, the Rough-Terrain Quaduped Robot, 2008.

[9] C. Semini. *HyQ Design and Development of a Hydraulically Actuated Quadruped Robot*. PhD thesis, Apr 2010.

[10] J.R. Rebula, P.D. Neuhaus, B.V. Bonnlander, M.J. Johnson, and J.E. Pratt. A Controller for the Littledog quadruped walking on Rough Terrain. In *IEEE International Conference on Robotics and Automation*, pages 1467–1473, April 2007.

[11] M. Ajallooeian, S. Pouya, A Sproewitz, and A.J. Ijspeert. Central Pattern Generators augmented with virtual model control for quadruped rough terrain locomotion. In *IEEE International Conference on Robotics and Automation*, pages 3321–3328, May 2013.

[12] Simon Rutishauser, A Sprowitz, L. Righetti, and A.J. Ijspeert. Passive compliant quadruped robot using Central Pattern Generators for locomotion control. pages 710–715, Oct 2008.

[13] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural networks*, 21(4), 2008.

[14] P. Arena. The central pattern generator: a paradigm for artificial locomotion. *Soft Computing*, 4(4):251–266, 2000.

[15] L. Righetti and A.J. Ijspeert. Programmable central pattern generators: an application to biped locomotion control. pages 1585–1590, May 2006.

[16] Vitor Matos and Cristina P. Santos. Omnidirectional locomotion in a quadruped robot: A CPG-based approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3392–3397, Oct 2010.

[17] G. Endo, J. Morimoto, J. Nakanishi, and G. Cheng. An empirical exploration of a neural oscillator for biped locomotion control. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE Int. Conf. on*, volume 3, pages 3036–3042 Vol.3, April 2004.

[18] Paolo Arena. A mechatronic lamprey controlled by analog circuits. In *MED'01 9th Mediterranean Conference on Control and Automation*. IEEE, June 2001.

[19] Bernhard Klaassen, Ralf Linnemann, Dirk Spenneberg, and Frank Kirchner. Biomimetic walking robot scorpion: Control and modeling. *Robotics and Autonomous Systems*, 41(23):69 – 76, 2002. Ninth International Symposium on Intelligent Robotic Systems.

[20] P. Arena, L. Fortuna, M. Frasca, and G. Sicurella. An adaptive, self-organizing dynamical system for hierarchical control of bio-inspired locomotion. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(4):1823–1837, Aug 2004.

[21] Shinkichi Inagaki, Hideo Yuasa, and Tamio Arai. {CPG} model for autonomous decentralized multi-legged robot systemgeneration and transition of oscillation patterns and dynamics of oscillators. *Robotics and Autonomous Systems*, 44(34):171 – 179, 2003. Best papers presented at IAS-7.

[22] Shinkichi Inagaki, Hideo Yuasa, Takanori Suzuki, and Tamio Arai. Wave {CPG} model for autonomous decentralized multi-legged robot: Gait generation and walking speed control. *Robotics and Autonomous Systems*, 54(2):118 – 126, 2006. Intelligent Autonomous Systems 8th Conference on Intelligent Autonomous Systems (IAS-8).

[23] A. Billard and A.J. Ijspeert. Biologically inspired neural controllers for motor control in a quadruped robot. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 6, pages 637–641 vol.6, 2000.

[24] Giorgio Brambilla, Jonas Buchli, and AukeJan Ijspeert. Adaptive four legged locomotion control based on nonlinear dynamical systems. In Stefano Nolfi, Gianluca Baldassarre, Raffaele Calabretta, JohnC.T. Hallam, Davide Marocco, Jean-Arcady Meyer, Orazio Miglino, and Domenico Parisi, editors, *From Animals to Animats 9*, volume 4095 of *Lecture Notes in Computer Science*, pages 138–149. Springer Berlin Heidelberg, 2006.

[25] J. Buchli, F. Iida, and A.J. Ijspeert. Finding resonance: Adaptive frequency oscillators for dynamic legged locomotion. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3903–3909, Oct 2006.

[26] K. Tsujita, K. Tsuchiya, and A. Onat. Adaptive gait pattern control of a quadruped locomotion robot. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 4, pages 2318–2325 vol.4, 2001.

[27] K. Tsujita, H. Toui, and K. Tsuchiya. Dynamic turning control of a quadruped robot using oscillator network. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2613–2618 Vol.3, April 2004.

[28] Miomir K. Vukobratovi. Contribution to the study of anthropomorphic systems. *Kybernetika*, 08(5):(404)–418, 1972.

[29] J.-I. Yamaguchi, A. Takanishi, and I. Kato. Development of a biped walking robot compensating for three-axis moment by trunk motion. In *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 1, pages 561–566 vol.1, Jul 1993.

[30] P. Sardain and G. Bessonnet. Forces acting on a biped robot. center of pressure-zero moment point. *Trans. Sys. Man Cyber. Part A*, 34(5):630–637, September 2004.

[31] A. Goswami. Foot rotation indicator (fri) point: a new gait planning tool to evaluate postural stability of biped robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 47–52 vol.1, 1999.

[32] Tsungnan Lin, B.G. Horne, P. Tino, and C.L. Giles. Learning long-term dependencies in NARX recurrent neural networks. *Neural Networks, IEEE Transactions on*, 7(6):1329–1338, Nov 1996.

[33] Grant P. M. Chen S., Billings S. A. Non-linear system identification using neural networks. In *Int. Journal of Control*, volume 51 of *Lecture Notes in Control and Information Sciences*, pages 1191–1214. Taylor and Francis, 1990.

[34] Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies, 1996.

[35] S. A Billings. *Nonlinear system identification : NARMAX methods in the time, frequency, and spatio-temporal domains.* John Wiley and Sons Ltd, 2013.

[36] Oliver Nelles. Neural networks with internal dynamics. In *Nonlinear System Identification*, pages 645–651. Springer Berlin Heidelberg, 2001.

[37] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[38] David E. Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation. chapter Backpropagation: The Basic Theory, pages 1–34. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[39] Raúl Rojas. The backpropagation algorithm. In *Neural Networks: A Systematic Introduction*, chapter 7. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[40] N. Hogan. Impedance control: An approach to manipulation. In *American Control Conference, 1984*, pages 304–313, June 1984.

[41] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404 vol.2, Apr 1991.

[42] F. Arambula Coso and M.A. Padilla Castaeda. Autonomous robot navigation using adaptive potential fields. *Mathematical and Computer Modelling*, 40(910):1141 – 1156, 2004.

[43] P. Krishnamurthy and F. Khorrami. Godzila: A low-resource algorithm for path planning in unknown environments. *Journal of Intelligent and Robotic Systems*, 48(3):357–373, 2007.

[44] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[45] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[46] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[47] Alberto Demichelis. Squirrel, the programming language, 2011.

[48] G. Bradski. *Dr. Dobb's Journal of Software Tools*.

[49] Beman Dawes. Boost C++ Libraries, 2005. [Online; http://www.boost.org/].

[50] wxWidgets, Cross-Platform GUI Library, 2015. [Online; https://www.wxwidgets.org/].

[51] P.-B. Wieber. Holonomy and nonholonomy in the dynamics of articulated motion. In Moritz Diehl and Katja Mombaur, editors, *Fast Motions in Biomechanics and*

Robotics, volume 340 of *Lecture Notes in Control and Information Sciences*, pages 411–425. Springer Berlin Heidelberg, 2006.

[52] Russel Smith. Open Dynamics Engine, 2008. [Online; http://www.ode.org/].

[53] Kiyotoshi Matsuoka. Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological Cybernetics*, 52(6):367–376, 1985.

[54] J.J. Collins and Ian Stewart. Hexapodal gaits and coupled nonlinear oscillator models. *Biological Cybernetics*, 68(4):287–298, 1993.

[55] Meng Yee (Michael) Chuah Park, Hae-Won and Sangbae Kim. Quadruped bounding control with variable duty cycle via vertical impulse scaling. 2014.

[56] Y. Fukuoka, H. Yasushi, and F. Takahiro. A simple rule for quadrupedal gait generation determined by leg loading feedback: a modeling study. *Sci. Rep.*, 5, 2015.

[57] Luiz Castro, Cristina P. Santos, Miguel Oliveira, and Auke Ijspeert. Postural Control on a Quadruped Robot Using Lateral Tilt: A Dynamical System Approach. In Herman Bruyninckx, Libor Peuil, and Miroslav Kulich, editors, *European Robotics Symposium 2008*, volume 44 of *Springer Tracts in Advanced Robotics*, pages 205–214. Springer Berlin Heidelberg, 2008.

[58] Jia-wang Li, Chao Wu, and Tong Ge. Central pattern generator based gait control for planar quadruped robots. *Journal of Shanghai Jiaotong University*, 19(1):1–10, 2014.

[59] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1620–1626 vol.2, Sept 2003.

[60] Katie Byl, Alec Shkolnik, Sam Prentice, Nick Roy, and Russ Tedrake. Reliable Dynamic Motions for a Stiff Quadruped. *International Symposium on Experimental Robotics*, pages 319–328, 2009.

[61] A. Takanishi, M. Tochizawa, T. Takeya, H. Karaki, and I. Kato. Realization of dynamic biped walking stabilized with trunk motion under known external force.

In KennethJ. Waldron, editor, *Advanced Robotics: 1989*, pages 299–310. Springer Berlin Heidelberg, 1989.

[62] R. Kurazume, T. Hasegawa, and K. Yoneda. The sway compensation trajectory for a biped robot. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 1, pages 925–931 vol.1, Sept 2003.

[63] Roberto Battiti. First and second-order methods for learning: between steepest descent and newton's method. *Neural Computation*, 4:141–166, 1992.

[64] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis. Improving the convergence of the backpropagation algorithm using learning rate adaptation methods. *Neural Comput.*, 11(7):1769–1796, oct 1999.

[65] G. Bradski and A. Kaehler. *Learning OpenCV*, chapter Image Morphology. OReilly Media, 2008.

[66] Niloy J. Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 322–328, New York, NY, USA, 2003. ACM.

[67] Edward Castillo, Jian Liang, and Hongkai Zhao. Point cloud segmentation and denoising via constrained nonlinear least squares normal estimates. In Michael Breu, Alfred Bruckstein, and Petros Maragos, editors, *Innovations for Shape Analysis*, Mathematics and Visualization, pages 283–299. Springer Berlin Heidelberg, 2013.

[68] J.Z. Kolter, Youngjun Kim, and A.Y. Ng. Stereo vision and terrain modeling for quadruped robots. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1557–1564, May 2009.