CHAPTER I

Introduction

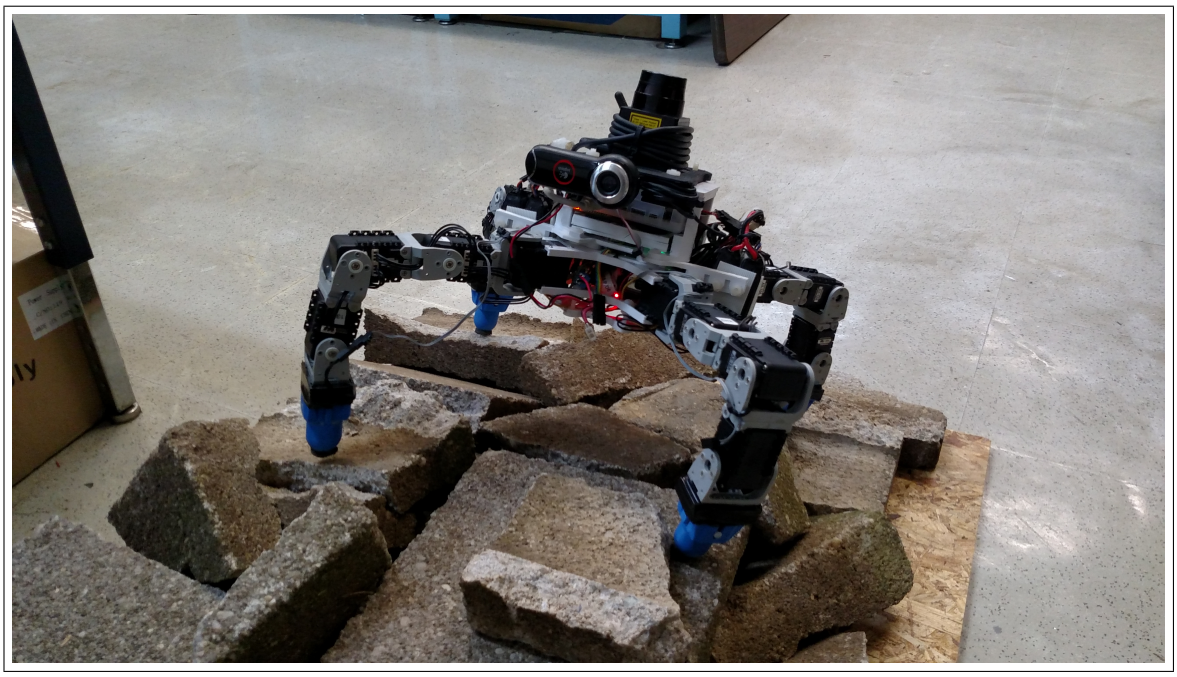

Figure 1: The BlueFoot Quadruped Robot

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 16 actuated degrees of freedom to allow for the execution of a wide range of body and leg articulations. Moreover, BlueFoot's range of articulation allows it to take on a large range of poses during motion. This level of dexterity grants the BlueFoot platform the several notable abilities. For one, it can overcome raised terrain, allowing it to traverse uneven terrain. Additionally, BlueFoot can articulate (*i.e.*, pitch and yaw) its on-board vision sensors array, which is attached to its main trunk, by reposing its body through 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 controller considers the systems kinematic model; and involves 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 gaiting technique which is augmented with a foothold controller along the same lines as [11] and [12]. Additionally, active platform stabilization is performed via a zero-moment point (ZMP) based body placement controller which stabilizes the system during arbitrary gaiting sequences. The controllers presented here make use of virtual-forces to drive system reference commands and make significant use of the system's forward kinematic model for the purpose estimating the positions of the robot's joints and feet. 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 features as target-points to track; and 3D point-cloud processing routines for surface reconstruction and foot-hold planning.

1.1 Central Pattern Generators for Gait Control

As previously mentioned, BlueFoot's core gaiting routine relies on the utilization of artificial CPG's, which are inspired from biological neural networks which generate

rhythmic motions [13]. [14] describes CPGs as a form of self-organizing cellular neural network, and also explains the a the role of limited feedback in CPG's. Infact, a key feature of these networks is that they can act without explicit feedback inputs, or even without directions from a higher-level command unit, such as a brain. Instead, signals emanating from independent motor units and feedback gathered from sensory neurons are utilized to trigger or inhibit a sequence of successive, self-coordinated motor operations. The activation sequences performed by a series of neural unit combine in phase-locked loops create cyclic motion patterns. In robotics, biological CPG's have inspired an artificial counterpart in which neural units are represented by multi-state unit-oscillator. The dynamics of these unit oscillators are coupled with other oscillators within the artificial CPG network. Typically, the CPG network is implemented on a controller, which numerically integrates the dynamics each neural oscillators. The output states of each unit-oscillator are used to drive selected degrees of freedom of a robot system through actuator reference-command signals. Oscillator outputs could also be used for planning periodic motions in the robot's task space, which are then translated into the joint-space via an inverse kinematics mapping, as is done in BlueFoot's gait control routine. The motions produced using the dynamical outputs of each unit-oscillator are usually coordinated through the careful tuning of oscillator coupling, which incurs particular phase offsets between the individual limit-cycles. This stable coordination between oscillators is what allows the CPG to be applied to the performance of a higher-level motor task, such as walking.

Studies dealing specifically the application of CPG's to multi-legged robot gaiting (specifically quadruped, hexapods and octopodal robots) have been carried out by [**?**, 15–23]. In particular, [13] states that the attractiveness of CPG's in the control of legged robot locomotion lies in a resulting ability to decouple robot motor control, *i.e.*, walking, from higher-level planning. Additionally, CPG's offer an effective way for smoothly switching between gaiting patterns, *e.g.*, walking, trotting, or pacing, by simply modifying only a few control parameters. As a result, the use of CPG's greatly reduces the dimensionality of the gaiting control problem by generating coordinated motions which can be modified to yield different overall motion patterns without explicit modification of each degree of freedom employed in gait execution.

An important aspect of the CPG-based gait design applied to the BlueFoot platform is the incorporation of feedback mechanisms which modify the aforementioned CPG parameters. The use of feedback to modify the CPG network for the purpose of improving

gaiting stability was guided, in part, by the work of [6, 24]. Namely, BlueFoot's CPG based gait generation incorporates inertial feedback signals into its CPG mechanism by using them to modify oscillator amplitudes and modulate unit-oscillator frequencies. Additionally, instead of using a CPG to control the outputs of individual quadruped joints, CPG outputs are mapped to stepping trajectories and foot-motion execution patterns. This approach yields a gaiting technique which combines the conveniences of CPG-based gaiting with the heightened control precision of explicit foot-step planning in the robot task-space. Moreover, foot-placement is explicitly prescribed via a separate planning mechanism which decoupled from the CPG gait controller. This method allows a CPG-based motion generation method to be applied gaiting over varying terrains.

Because CPG-based gaits are inherently open-loop motion control routines, a combination of axillary mechanisms must be used in concert with the base CPG gait controller in order to ensure system stability during gaiting. The incorporation of feedback signals to modified CPG parameters aids in achieving this to some degree, but is usually insufficient for stable walking over largely uneven terrains.

Additionally this method would require 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. In particular, BlueFoot's core stabilization routines make use of a concept formally named *artificial synergy synthesis* in which gaiting is carried out independently of a stabilization control. Namely, body stabilization is performed by a restricted set of the robot's degrees of freedom while gaiting is carried out independently by the remaining [25, 26]. In an original implementation of this technique, adaptations to trunk motion are utilized to stabilize the overall motion of the robot utilizing a ZMP-based approach, as is done here, while gaiting is controlled by a fixed-motion routine. Here, body and foot-placement are both controlled dynamically, but still independently. The outputs of each controller, which specify body and foot placement in the robot task space, respectively, are combined using the robot's inverse kinematics solution, which generates a final set of joint references.

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 the 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 [27].

The concept of ZMP and its application to legged robotics was originally introduced by [25] and expanded upon in [28], both of which describe ZMP theory towards use in the control of biped robots.

Formally, the ZMP, p_{ZMP}, can be defined using a formulation for the CoP wherein the moments about τ_x and τ_y, the 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, as shown in [5]. This expression is derived from an inspection of the dynamics which govern 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 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

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

where

$$C_{ZMP} = \text{conv}\,(p_{1,e}, p_{2,e}..., p_{N,e}) \tag{1.4}$$

with conv$(*)$ defining a convex hull from the input points, $(*)$, and is used to represent the solution space of p_{ZMP}. Moreover, 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 $\dot{L} = 0$, we defined a condition 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 stability condition:

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

where $\epsilon \ll 1$ is a small bounding constant. 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, thus minimizing $\left\|\dot{L}\right\|$ so as to avoid tipping.

1.3 Trunk stabilization

In addition to the aforementioned task-space controllers, a learning controller, which features the use of a NARX neural network is used to aid trunk stability has been studied. 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 such 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 [29–32]. Moreover, the NARX-NNs is a natural fit for a problem of this nature 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 [33].

The NARX-NN is trained to capture the effects of forces and moments and dynamical couplings that act on the trunk so that an appropriate torque input to the joints is computed to reduce such 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 [34, 35]. 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 summarized in matrix-vector form in (adapted from [36]) as follows:

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

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

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

For the trunk-leveling controller being described, a symmetric sigmoid activation function is used, making each $\sigma_j^i(y_j^i) \equiv \tanh(y_j^i) \in [-1, 1]$. Hence the gradient $\nabla_y \sigma^i(y^i)$ is defined as follows:

$$\nabla_y \sigma^i(y^i) = \begin{bmatrix} \sigma_1^1(y^i) & 0 & \dots & 0 \\ 0 & \sigma_1^2(y^i) & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & \sigma_1^{N_I^i}(y^i) \end{bmatrix} \left(\vec{1}_{N_I^i \times 1} - \sigma^i \left(y^i \right) \right) \tag{1.7}$$

given the derivative properties of the $\tanh(*)$ function.

The success of this learning mechanism, as it applies to the presented controller, 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

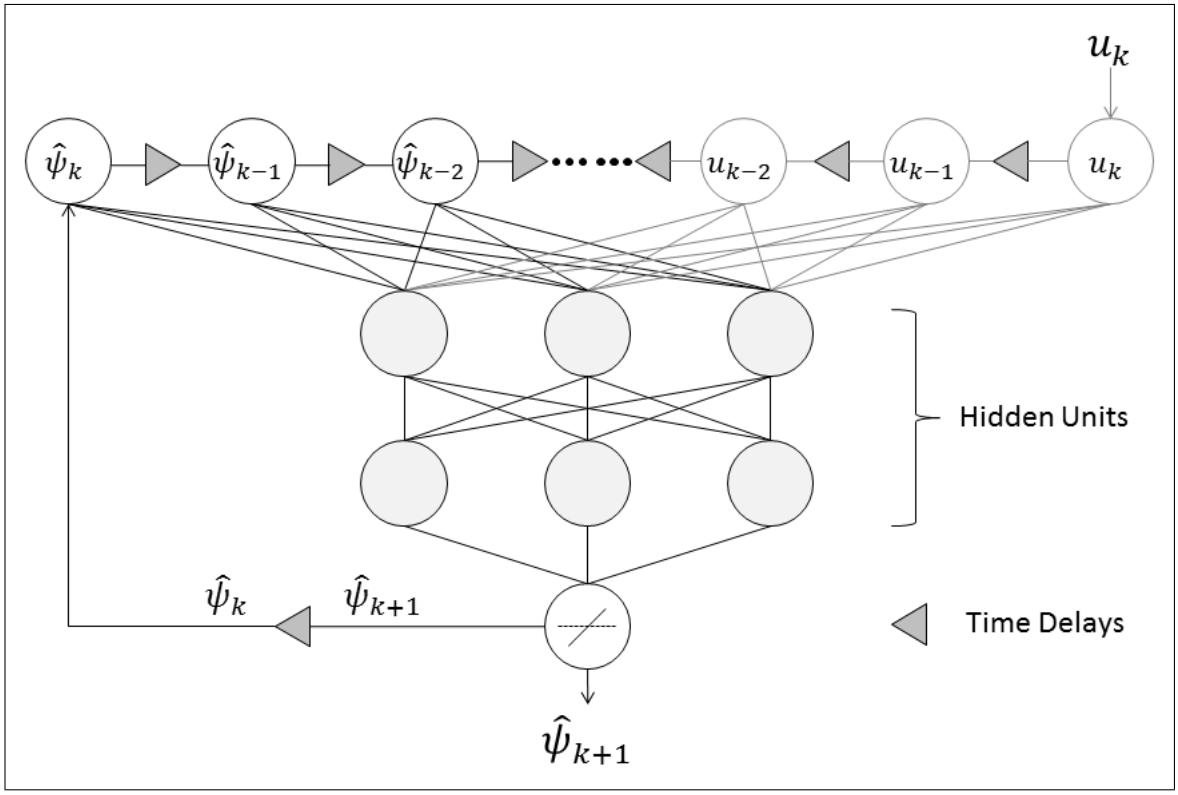

Figure 2: Parallel NARX-network model with a linear output layer.

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 Navigation and 3D Reconstruction

One method that has been selected for navigating the BlueFoot platform over flatland is a potential-fields control approach, described in [**?**] for the purpose of controlling robotic manipulators, and analyzed in-depth in [**?**]. In particular [**?**] presents shortcomings of this approach. Nonetheless, a potential fields navigation methods offers a reflatively simple and intuitive approach to robot navigation and fits well into mobile robotic tasks which involve "wandering" type navigation over flat regions, in which the robot has yet to aquire any knowledge of surrounding obstacles. In this thesis, a potential fields navigation approach is used to navigate BlueFoot in unknown regions, and is coupled with camera-based feature tracking to guide the robot towards potential areas of interest within an unknown, immediate space.

The potential fields approach is used in mobile robot navigation by moving the

robot according to a guiding virtual force-vector, F_{nav} [**?**, 58]. This vector is comprised of a sum of virtual repuslive forces, F^- (typically generated range-sensor data), and virtual attractive forces, F^+, which pull the robot towards known goals. Thus F_{nav} formally defined as:

$$F_{nav} = F^+ + F^- \tag{1.8}$$

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

$$\begin{aligned} d_k &= p_{POI,k} - p_{robot} \\ F_c &= \alpha_F \sum_k \left(f(\|d_k\|) \frac{d_k}{\|d_k\|} \right) \end{aligned} \tag{1.9}$$

where p_{poi} and p_{robot} are 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 "energy" factor with respect to a scalar distance argument $(*)$; and α_F is a scaling parameter which is positive for when POIs represent goals and negative when POIs represent obstacles to avoid. The potential function and force-scaling factors are designable for particular applications. For BlueFoot's navigation scheme, attractive and repulsive forces are generated using a single, consolidated forcing function which is used to guide through an environment even when a goal is not specified before hand.

The aforementioned potential field navigation method is used, primarily, to handle BlueFoot's flatland navigation. Navigation over rough terrain requires a number of other considerations about the terrain of the enviroment itself, as opposed to simple POIs which can be resolved by observing the enviroment around the robot in 2D. Thus, surface reconstruction plays a large roll in the process of navigation over irregular terrain. This thesis will present two methods of surface reconstruction, which will serve as prelimaries for rough-terrain navigation and planning in futurer work with the BlueFoot platform.

1.5 Overview of Thesis

This thesis will first detail the major hardware components; design considerations; and construction of the BlueFoot platform. Next, the software and processing architecture used to control the BlueFoot platform will be described. Thereafter, the kinematic and dynamical model of the BlueFoot system will be described, followed by control routines which are presently implemented to gait, stabilize, and navigate the BlueFoot platform. The final section of this thesis will contain concluding statement about the

system design and control, including remarks about possible future directions of study related to the BlueFoot platform and legged robotics as a whole.

CHAPTER II

Hardware and Design

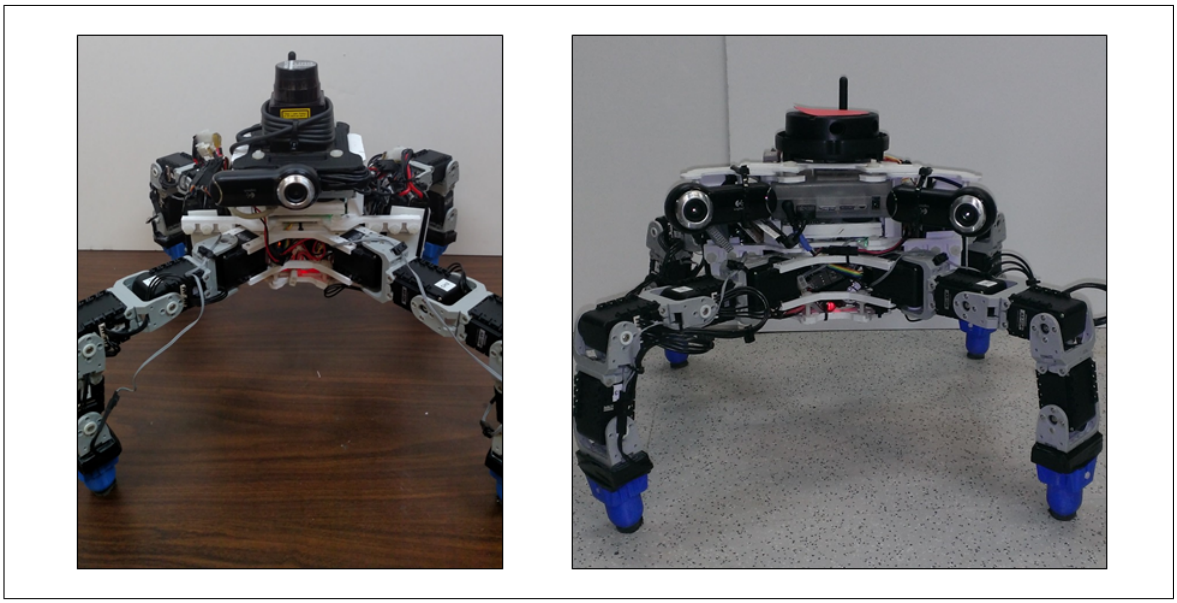

Figure 3: The BlueFoot Quadruped Robot: single-camera configuration *(left)*; stereo-camera configuration *(right)*

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.

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

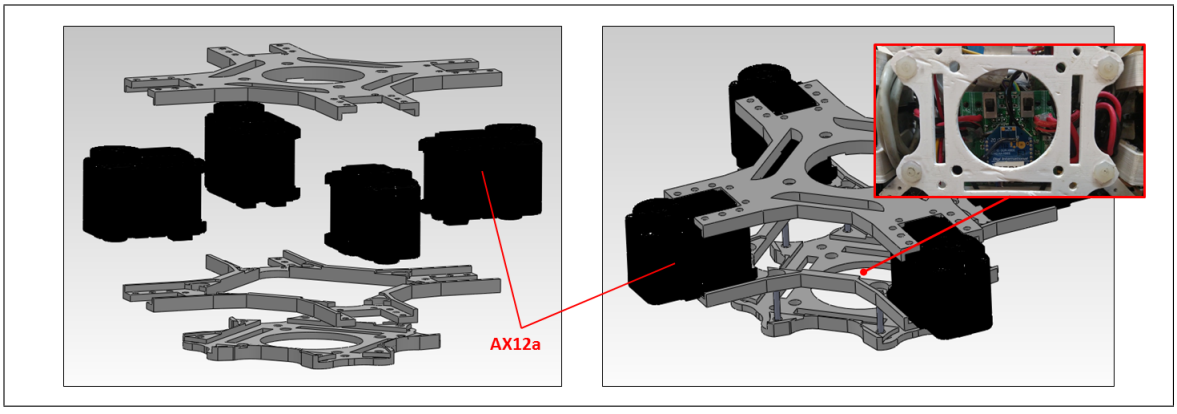

Figure 4: Root section of trunk. Call-out in top-right shows main-switch access through the bottom of the root module.

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.

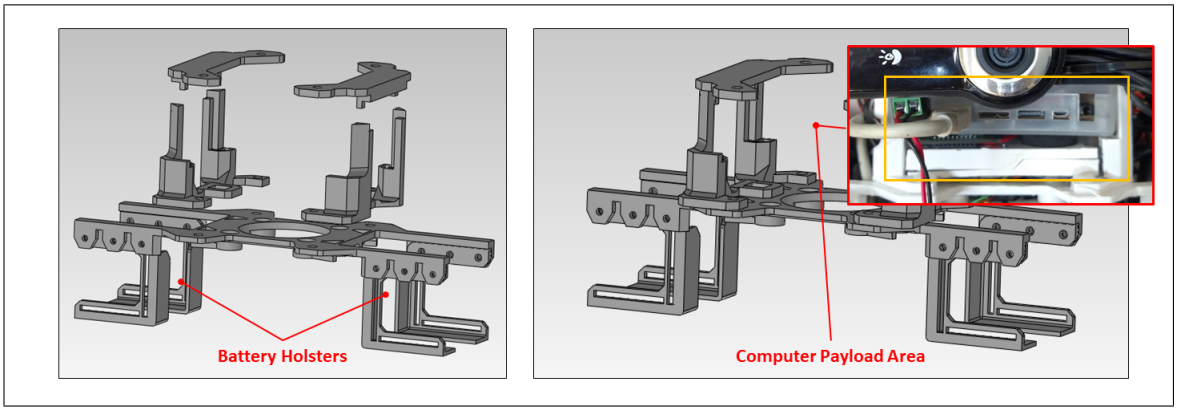

Figure 5: Main section of trunk. Call-out in the top-right shows how the ODROID-XU and AutoPilot computers fit within the module.

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.

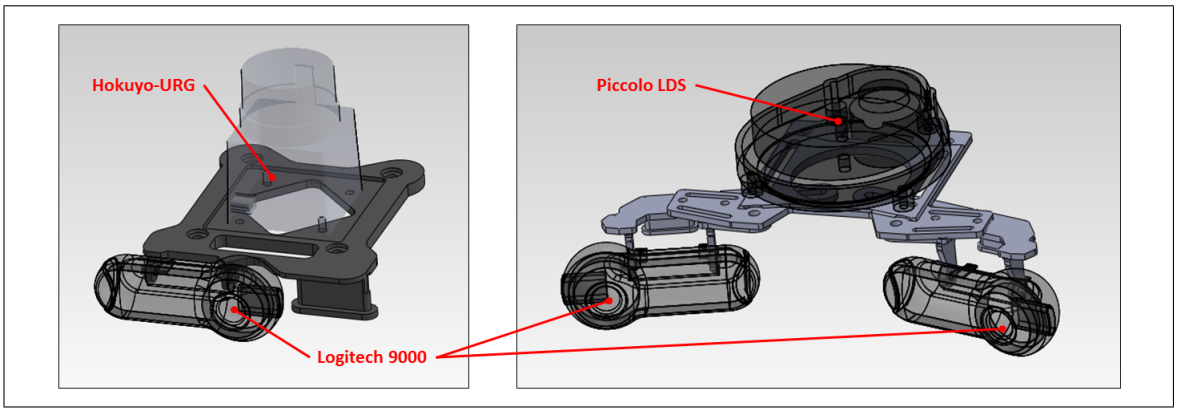

Figure 6: Head section of trunk. Monocular camera configuration with Hokuyo-URG *(left)*; and Stereo configuration with PLDS *(right)*.

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

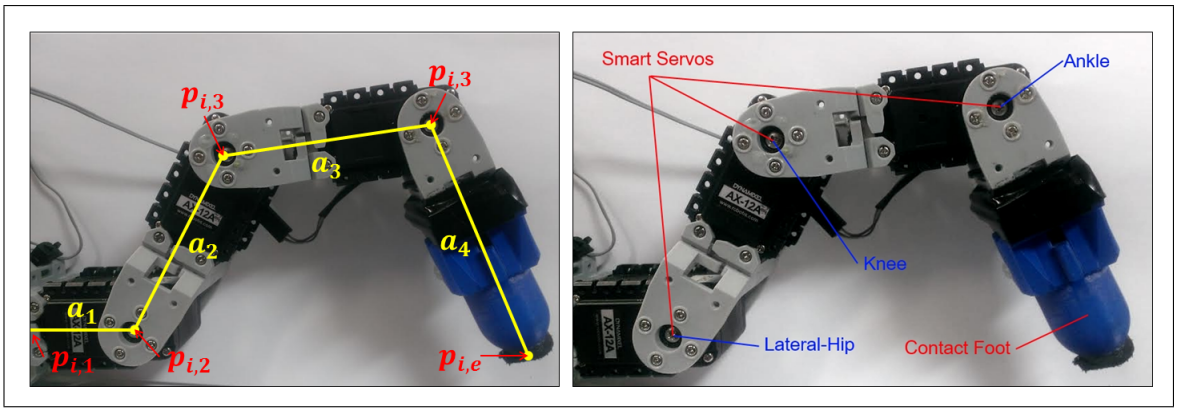

Figure 7: Closeup of BlueFoot's leg. *(left)* shows effective link lengths and the location of defined joint positions.

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.

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).

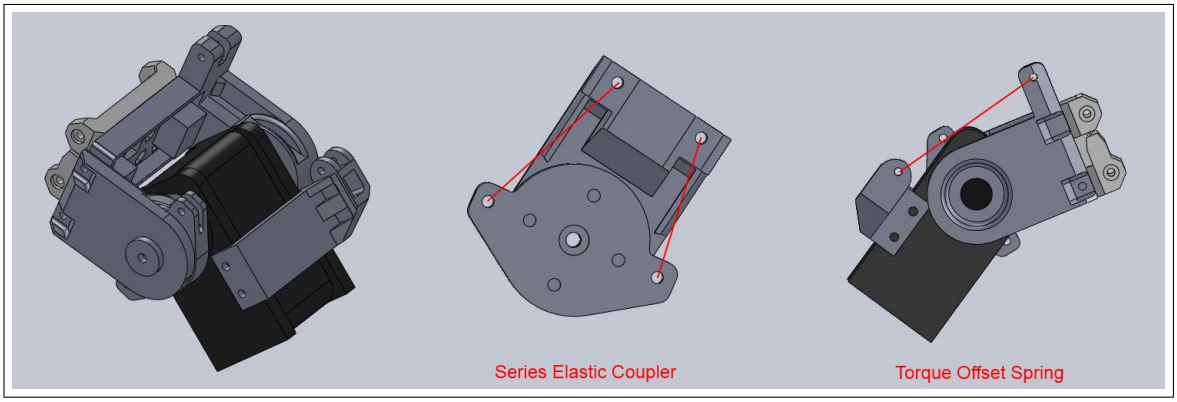

Figure 8: Series elastic brackets.

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

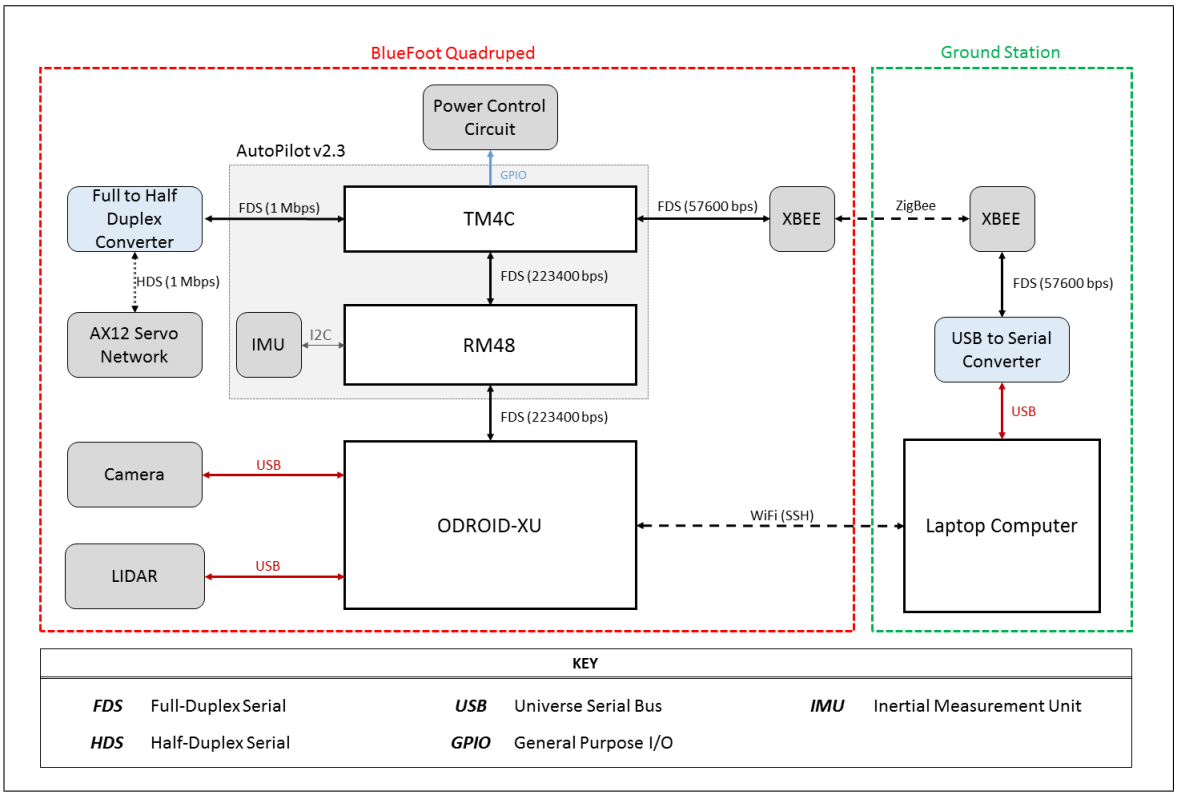

Figure 9: BlueFoot device networking diagram.

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.

Port$_A$	**Source**$_A$	**Port**$_B$	**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.

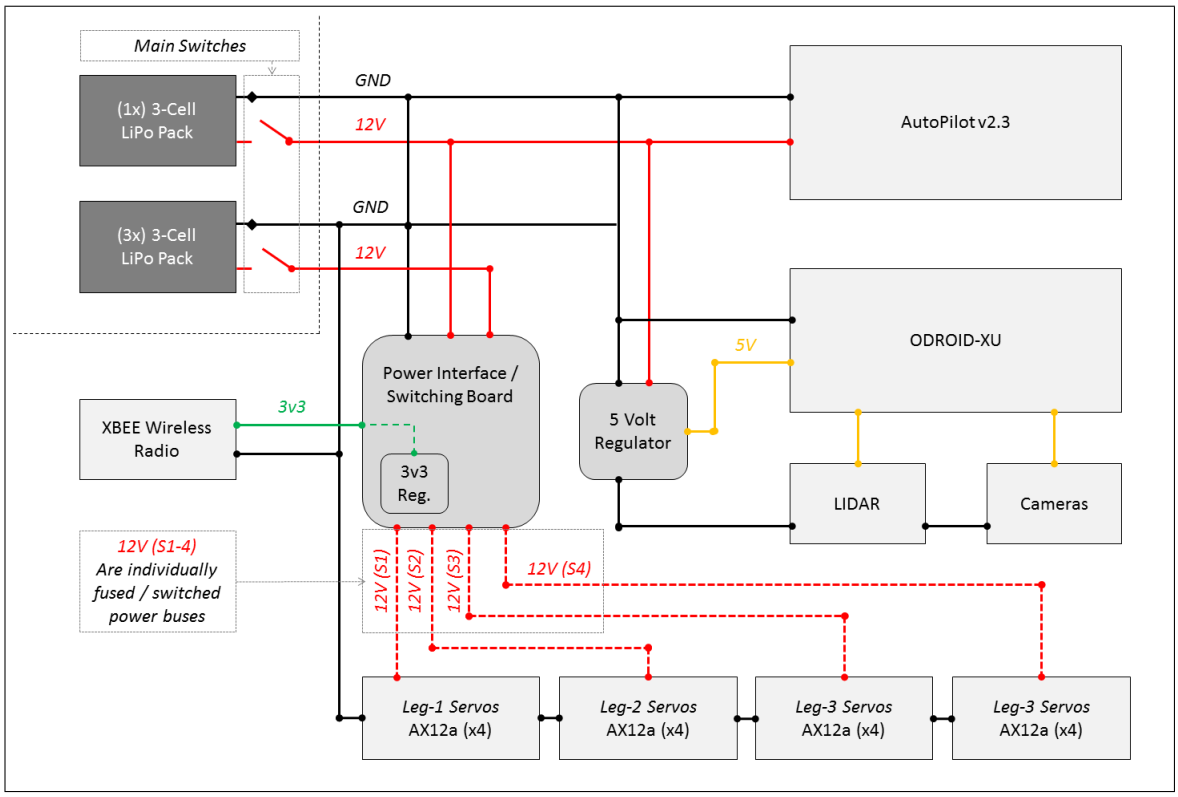

Figure 10: BlueFoot power routing.

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. Moreover, each of these processors handles specific subsets of operations essential to the macro-system. This distribution of system tasks across several core operating units allows 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 was designed such that core programs could be readily offloaded to physically separate computing modules. Each of these control modules handles thier 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 the software collective spanning over the RM48 and TM4C processors on-board 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 (ODROID) 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 communication lines which enter the TM4C processor and the ODROID computer. The ground-station also interfaces with the UB over an SSH connection. This secondary wireless connection is used, mainly, for on-board data-logging configurations.

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 the specifics of 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 a general description of 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 a safety/communications routing co-processor within the overall system. Within its main program loop, the TM4C polls the system's main battery voltage via ADC interface routines; handles transmit and receive

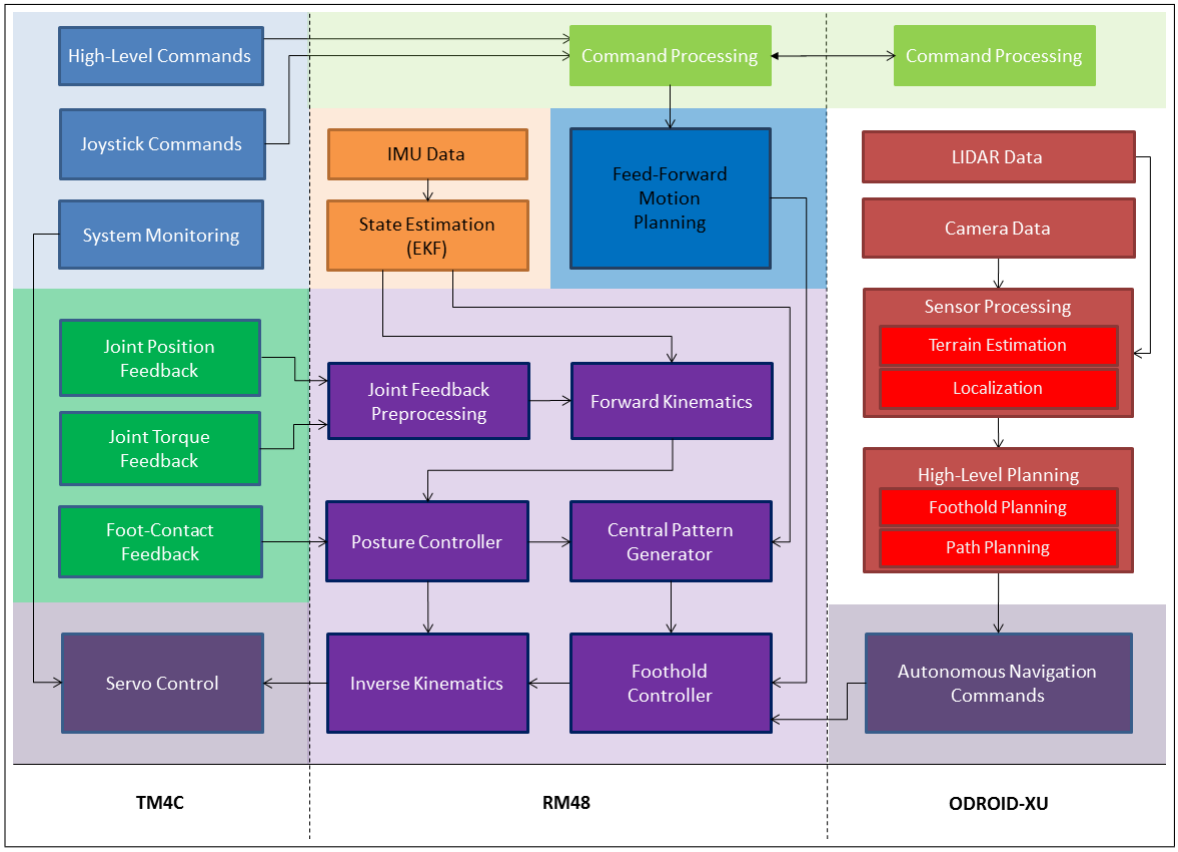

Figure 11: BlueFoot's on-board processes/signals and their relationships.

(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 directly interfaced with two dual-channel power switching IC's and is used to control power supply to each leg 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/or 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 controllers on-board each smart-servo. Administering servo commands and collection 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 loop frequency to roughly 50 Hz.

Thus, it is particularly important that this task is offloaded to this processor, as its 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 collected IMU sample is passed along to an extended Kalman Filter routine, which generates a trunk orientation estimate, $\hat{\theta}_b$ in the world frame, O_0.

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 gait control via 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 leveling procedures, and gait-stabilization 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 exectued at all times when the legs are engaged for the purpose of issuing servo position commands, given desited task-space configurations for body and feet positions, which will be more formally defined in Chapter IV. The RM48 also maintains BlueFoot's forward kinematic model (specifically, foot position relative to the trunk), which relies on an 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), 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 [37]. 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 naviga-

tion, among other high-level command itams. Scripting commands are passed from the ground station (via terminal) and routed through the TM4C to the RM48, where they are finally evaluated.

ODROID-XU (Upper Brain)

The ODROID-XU runs software upon a Debian (Linux) operating system distribution "Jessie." The use of an Linux operating system extends itself to a number of programmatic conveniences, such as to ability to run several tasks in parallel threads. Inbuilt USB drivers are used in functions which 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 control; as well as LIDAR scan frames. The ORIOID uses these sensor inputs, in conjunction with orientation estimates and inertial data passed from the RM48 to perform several navigation-related tasks (*e.g.*, potential fields, mapping, localization, terrain-reconstruction). These tasks will be described in more detail in Chapter VI.

The ODROID 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 frames are utilized in potential fields-based navigation tasks. These navigation modes also incorporate camera data for the purpose of goal-targeting (where the goal is typically an object of particular shape or color). Image processing and image feature detection is run as a separate process on the ODROID, which is incorporated with the aforementioned processed sensor data to produce a set of forward and turning velocity commands, v^r and ω^r, respectively; as well as foot-hold positions generated from step-planning algorithms. 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 [38–40]. Software written for this platform was generated using a mixture of C++ and Python.

As previously mentioned, the ODROID can handle a limited set user-command on its own, which are administered directly to the ODROID from an SSH terminal on the ground-station computer. These commands include core-program start-ups and data-logging configurators. Essentially, the ODROID's software core is designed as a completely independent software module which replaces the roll a human director, as it handles the bulk of the systems high-level planning and navigation tasks. Moreover, if

the ODROID is removed from the BlueFoot system, the system can still be operated via remote-control heading commands provided from human operator (*i.e.*, ground-station joystick control).

3.1.2 Inter-processor Communication

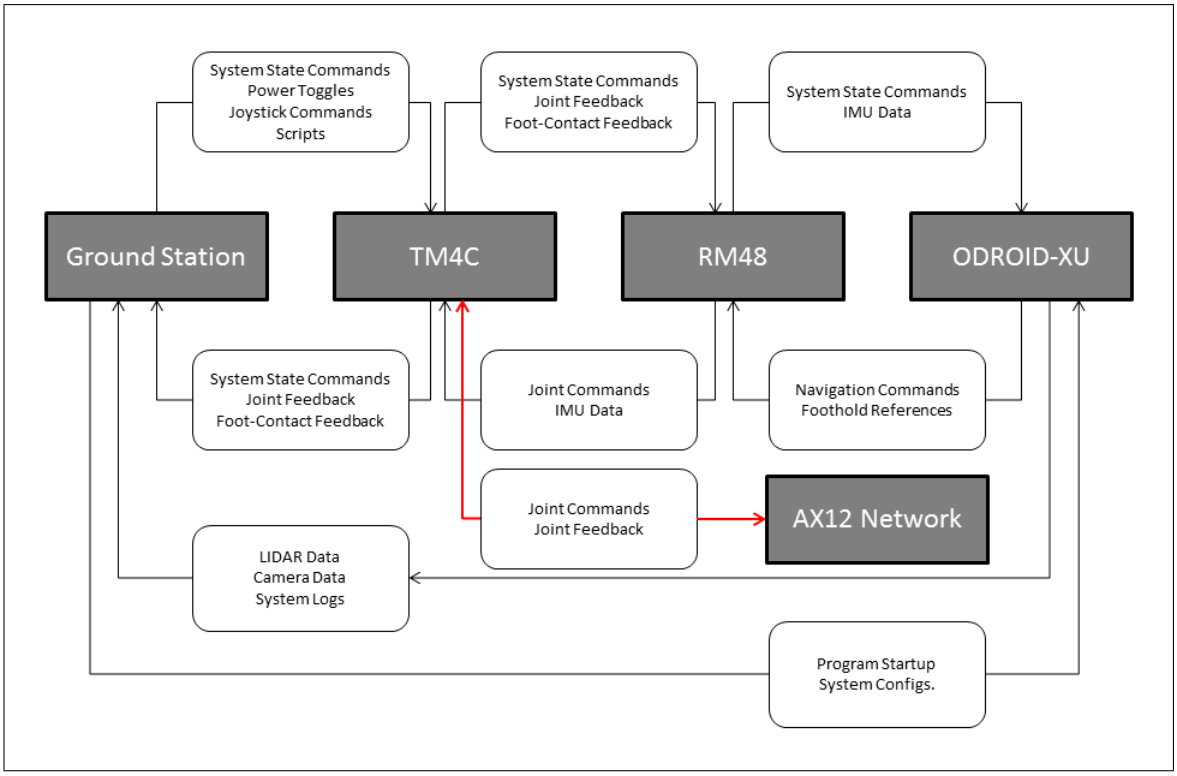

Figure 12: Communication flow between processors on the BlueFoot Platform.

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/or joystick controllers, are generated from a ground-station computer. Packets (without padding) sent from the ground station to the BlueFoot robot (TM4C) are composed as shown in Table 5:

Every packet issued using the EXI protocol has a 4-byte (32-bit) header. As part of the internal system protocol, every packet sent between system nodes contains a "Master Status Vector", which is comprised of a fixed length, 7-byte sequence of essential system information/control items. The "Master Toggles" section (first 8-bits after header) enumerates major systems states, including *On-line, Standby, Off-line* and *Suspended*

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.

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, foward 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 items generated on board the robot, and appear as shown in Table 6: The *Joint Pos. FB* (joint position

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.

feedback) element is composed of 16 2-byte sequences corresponding to the joint positions of read-back by 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. Like all system packets, 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 with corresponding joint-position commands for each of the 16 servo actuators. This field is also 2048 bits in length.

Packets sent from the RM48 to the ODROID contain additional dynamical-state fields for use in planning on the ODROID. State information is sent in the form a vectors with 32-bit, single precision floating-point elements. This set of information includes

trunk a orientation estimation, angular rate, and global position (generated from open-loop command integration), each of which are represented as 3-element vector; and foot-position estimates (four, 3-element vectors.) generated. These packets have a structure which is depicted in Table 7. Packets sent form the ODROID to the RM48 contain

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.

command items, such as forward velocity, turning rate and trunk-pose commands, as 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 to the RM48.

3.2 Ground Station

Ground-station software used for controlling the BlueFoot platform is generated in C++ using an open-source graphical user interface design library called *wxWidgets* [41]. *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 via an XML-based design specification file. Furthermote, this allows for UI reconfigurations without the need to change internal, back-end processes. In terms of the GUI backend, *wxWidgets* handles general UI signal processing, and allows for easy registration of user callbacks on events such as button presses. Additionally, *wxWidgets* provides an interface for collective joystick inputs, which are interpretted and sent to the BlueFoot as remote-control commands. *Boost*, a C++ utility library, is utilized to provide socket and serial-port IO handling. 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 UI sections provide interfaces for administering system master-state, opterating-state and power toggling; providing system commands; and scripting. Basic system state controls are used to periodically populate a fixed-structure, robot-command packet, shown in Table 5. This packet is used to refresh the robot operating state and manual navigation commands. Updates from the groundstation 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 ensure that system updates and commands have been recieved 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 coniditons
- foot contact feedback
- and joint position feedback.

This data is used to update several key portions of the UI. Firstly, 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 commands are used to update a visualization of the robot in real time. The foot-color of the vizualized BlueFoot robot changes from blue to red to represent foot-contact conditions. 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 foot/body trajectory planning, localization, and adaptation. In particular, inverse position and velocity solutions are used to prescribe joint-space commands from particular foot trajectories planned within the world coordinate frame. Additionally, forward kinematic solutions are utilized to localize the position of each foot using a combination of localized trunk position/orientation and joint position feedback. This section will fully describe the forward and inverse kinematic models which describe the BlueFoot platform and are 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 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 $\theta_b \in R^3$ and $p_b \in R^3$, 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),

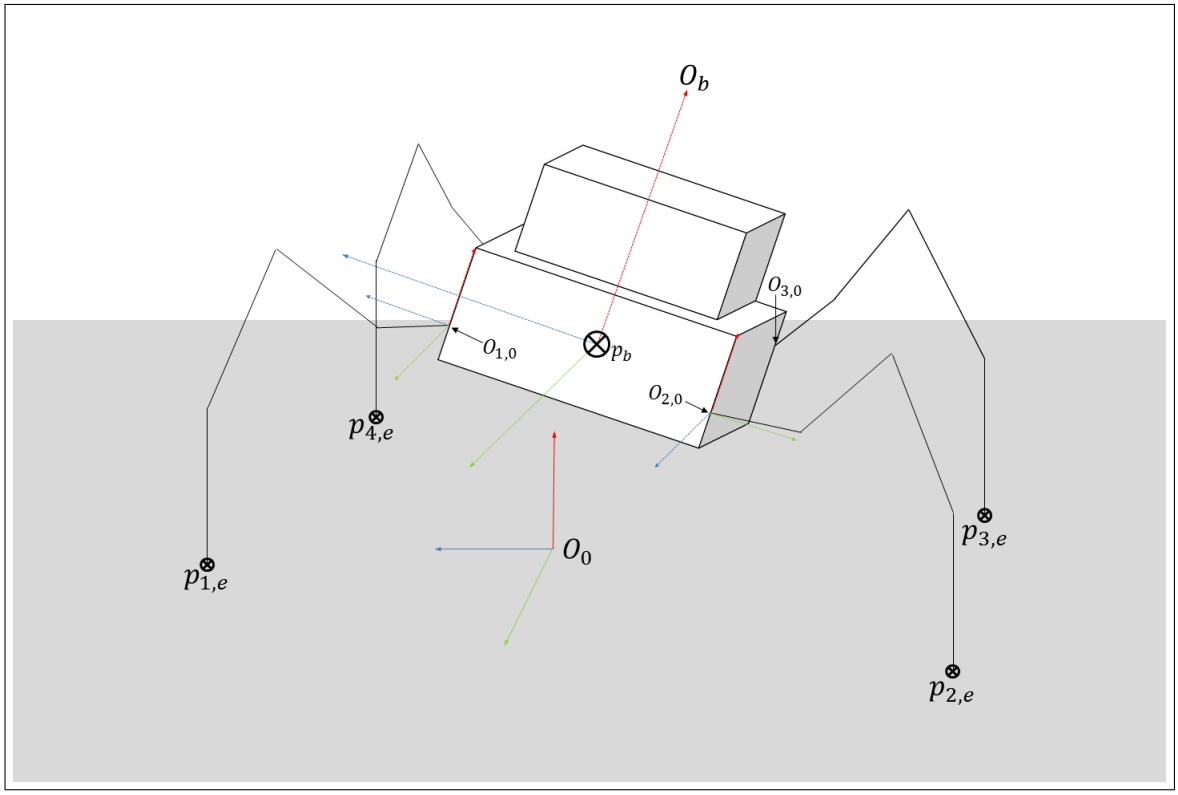

Figure 13: A visualization of BlueFoot's coordinate frame configuration.

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 com-

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.

puted explicitly as follows:

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

$$H^{i,0}_{i,2} = \left[\begin{array}{ccc|c} c_{1,i}c_{2,i} & -c_{1,i}s_{2,i} & s_{1,i} & c_{1,i}(a_{1,i} + a_2c_{2,i}) \\ s_{1,i}c_{2,i} & -s_{1,i}s_{2,i} & -c_{1,i} & s_{1,i}(a_{1,i} + a_2c_{2,i}) \\ s_{2,i} & c_{2,i} & 0 & a_2s_{2,i} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.4}$$

$$H^{i,0}_{i,3} = \left[\begin{array}{ccc|c} c_{1,i}c_{23,i} & -c_{1,i}s_{23,i} & s_{1,i} & c_{1,i}(a_{1,i} + a_2c_{2,i} + a_3c_{23,i}) \\ s_{1,i}c_{23,i} & -s_{1,i}s_{23,i} & -c_{1,i} & s_{1,i}(a_{1,i} + a_2c_{2,i} + a_3c_{23,i}) \\ s_{23,i} & c_{23,i} & 0 & a_2s_{2,i} + a_{3,i}s_{23,i} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.5}$$

$$H^{i,0}_{i,4} = \left[\begin{array}{ccc|c} c_{1,i}c_{234,i} & -c_{1,i}s_{234,i} & s_{1,i} & c_{1,i}(a_{1,i} + a_2c_{2,i} + a_3c_{23,i} + a_4c_{234,i}) \\ s_{1,i}c_{234,i} & -s_{1,i}s_{234,i} & -c_{1,i} & s_{1,i}(a_{1,i} + a_2c_{2,i} + a_3c_{23,i} + a_4c_{234,i}) \\ s_{234,i} & c_{234,i} & 0 & a_2s_{2,i} + a_3s_{23,i} + a_4s_{234,i} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.6}$$

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

$$E_p = [I_{3\times3}, 0_{3\times1}]$$
$$e_p = [0_{1\times3}, 1]^T.$$

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:

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

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 a Extended Kalman Filter (EKF), or one of many other orientation estimation methods (*i.e.*, complimentary filter, etc.). 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 often 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 relate $O_{b'}$ 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:

$$H_{b'}^{i,0} = \left[\begin{array}{c|c} R_{zyx}(\theta_b)R_z(\sigma_i) & R_{zyx}(\theta_b)\nu + p_b^{b'} \\ \hline 0 & 1 \end{array}\right] \tag{4.10}$$

which follows the same definitions in (4.1). The transformation defined in (4.10) is used as a direct replacement for the matrix $H_0^{i,0}$ in (4.8) and (4.9) in computing the corresponding particles, $p_{i,j}^{b'}$ and $p_{i,e}^{b'}$. Knowledge of the position of each respective

particle is useful for planning foot and body-placements which depend directly on the location other foot positions and the relative body location, 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:

$$\begin{aligned}
q_{i,1} &= \cos(i\pi)\left(\frac{\pi}{4} - \psi_i\right) \\
q_{i,2} &= \tan^{-1}\left(\frac{\zeta_{i,z}}{\sqrt{\zeta_{i,x}^2 + \zeta_{i,y}^2}}\right) \mp \cos^{-1}\left(\frac{a_3^2 - a_2^2 - \|\zeta_i\|^2}{2a_2\|\zeta_i\|}\right) \pm \pi \\
q_{i,3} &= \mp\cos^{-1}\frac{\|\zeta\|^2 - a_2^2 - a_3^2}{2a_2a_3} \\
q_{i,4} &= \gamma_i - q_{i,2} - q_{i,3}
\end{aligned} \tag{4.11}$$

where

$$\begin{aligned}
p_{i,e}^{i,0} &= E_p\left(H_{i,k}^0\right)^{-1}\begin{bmatrix} p_{i,e} \\ 1 \end{bmatrix} \\
\psi_i &\equiv \tan^{-1}\left(\frac{\left[p_{i,e}^{i,0}\right]_y}{\left[p_{i,e}^{i,0}\right]_x}\right) \\
\zeta_{i,x} &\equiv \left[p_{i,e}^{i,0}\right]_y \sin(\psi_i) + \left[p_{i,e}^{i,0}\right]_x \cos(\psi_i) - a_4\cos(\gamma_i) - a_1 \\
\zeta_{i,y} &\equiv \left[p_{i,e}^{i,0}\right]_y \cos(\psi_i) - \left[p_{i,e}^{i,0}\right]_x \sin(\psi_i) \\
\zeta_{i,z} &\equiv \left[p_{i,e}^{i,0}\right]_z - a_4\sin(\gamma_i).
\end{aligned} \tag{4.12}$$

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 $p_{i,e}^{i,0}$.

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 10) and has been performed with all feet fixed in a default stance (see 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.

	$p_b^{b'}$	$p_{1,e}^{b'}$	$p_{2,e}^{b'}$	$p_{3,e}^{b'}$	$p_{4,e}^{b'}$
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 $O_{b'}$.

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 $O_{b'}$.

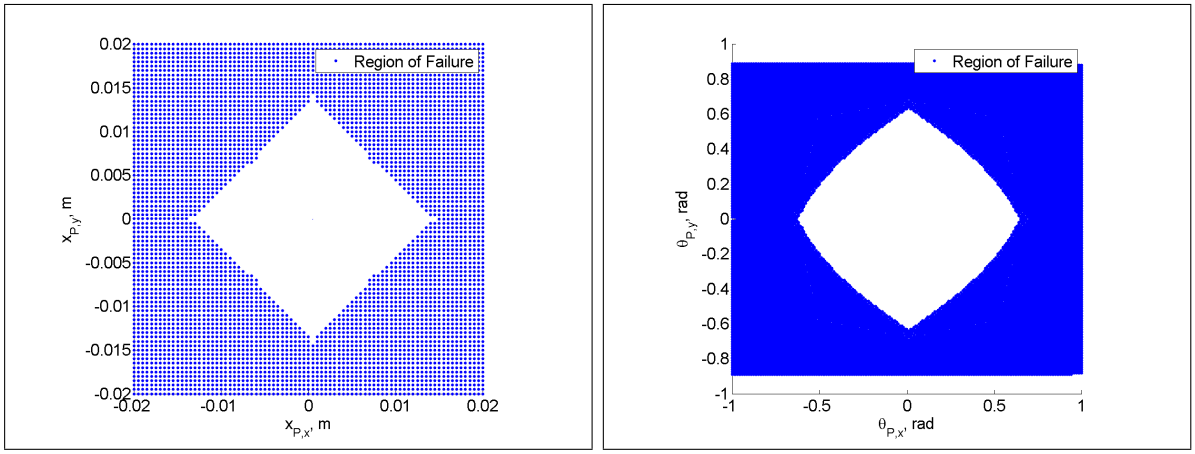

Figure 14: Range of motion for body's planar position (*left*) and body pitch and roll (*right*) while in the default stance.

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 $J_{i,e}^{i,0} \in R^{6\times 4}$ where

$$\dot{x}_{i,e}^{i,0} = J_{i,e}^{i,0}\dot{q}_i \tag{4.13}$$

with $\dot{x}_{i,e}^{i,0} \in R^6$ being stacked vector of translational and rotational velocities, $\dot{p}_{i,e}^{i,0} \in R^3$ and $\dot{\theta}_{i,e}^{i,0} \in R^3$, 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, $\dot{p}_{i,e} \in R^3$ and $\dot{\theta}_{i,e} \in R^3$, respectively, are known, the translation velocity of each i^{th} foot can be written with respect to frame $O_{i,0}$ by:

$$\dot{p}_{i,e}^{i,0} \equiv \left(R_{i,0}^0\right)^T \left(\dot{p}_{i,e} - \dot{p}_b - S\left(\dot{\theta}_b\right)\left(p_{i,e} - p_b - R_{i,0}^0\vec{o}_\nu\right)\right) \tag{4.14}$$

where $\vec{o}_\nu = [\nu, 0, 0]^T$, $R_{i,0}^0$ is the rotation-matrix component of the transformation $H_{i,0}^0$ defined in (4.1), and $S(*)$ is the standard 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:

$$\Theta^i \equiv \left(R_{i,0}^0\right)^T S\left(\dot{\theta}_{i,e} - \dot{\theta}_b\right) R_{i,0}^0 = S\left(\dot{\theta}_{i,e}^{i,0}\right) \tag{4.15}$$

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

$$\dot{\theta}_{i,e}^{i,0} \equiv \left[-\Theta_{1,2}^i, \Theta_{1,3}^i, -\Theta_{2,3}^i\right]^T \tag{4.16}$$

with $\Theta^i_{i,j}$ being the (i^{th}, j^{th}) element of the skew symmetric matrix Θ^i which results from (4.15).

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:

$$\begin{aligned} \dot{q}_i &\approx \left[J^{i,0}_{i,e}\right]^{\dagger}_{\Lambda_j} \dot{x}^{i,0}_{i,e} \\ \dot{q}_i &\approx \left(\left(J^{i,0}_{i,e}\right)^T J^{i,0}_{i,e} + \Lambda_J\right)^{-1} \left(J^{i,0}_{i,e}\right)^T \dot{x}^{i,0}_{i,e} \end{aligned} \tag{4.17}$$

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 pseuedo-inversion to avoid a non-smooth set of solutions $\dot{q}_i$. Degenerate solutions for $\dot{q}_i$ exist a particlar 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:

$$M(\eta)\ddot{\eta} + C(\eta, \dot{\eta})\dot{\eta} + G(\eta) + \Delta H = \tau + J^T(\eta) f_{ext} \tag{4.18}$$

where $M(\eta)$, $C(\eta, \dot{\eta})$, $G(\eta)$ and $J(\eta)$ represent the system mass matrix, Coriolis matrix, gravity matrix and Jacobian, respectively [42]. ΔH has been included as a lump term to account for dynamical uncertainties, such as friction or unmodeled coupling effects. Additionally, $f_{ext} = [f^T_{1,ext}, f^T_{2,ext}, f^T_{3,ext}, f^T_{4,ext}]^T \in R^{24}$ 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: $\eta = [p^T_b, \theta^T_b, q^T]^T$ 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 (4.18), can be realized in a general, compact, state-space form by:

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= M^{-1}(z_1)(\tau + \Phi(z_1, z_2, f_{ext})) \\ \Phi(z_1, z_2, f_{ext}) &= J^T(z_1) f_{ext} - C(z_1, z_2) z_2 - G(z_1) - \Delta H \end{aligned} \tag{4.19}$$

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:

$$\begin{aligned} z_{1,k+1} &= z_{1,k} + (e_{1,k}^{\Delta_s} + z_{2,k})\Delta_s \\ z_{2,k+1} &= z_{2,k} + M_{1,k}^{-1}(e_{2,k}^{\Delta_s} + \tau_k + \Phi_k)\Delta_s \\ t &= \Delta_s k \end{aligned} \tag{4.20}$$

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:

$$\tau_q = k_s(q^r - q) \tag{4.21}$$

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

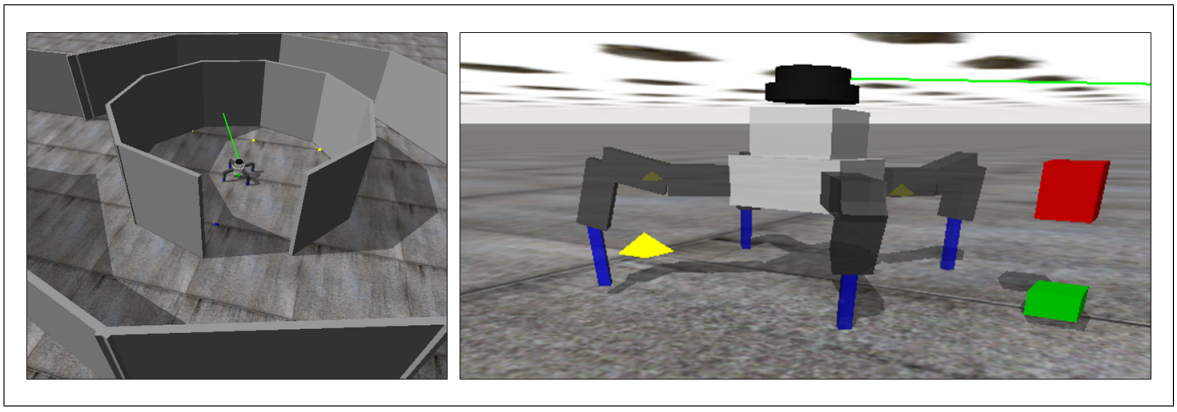

Figure 15: Visualization of the BlueFoot simulator.

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) [43], 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 (4.21). 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-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

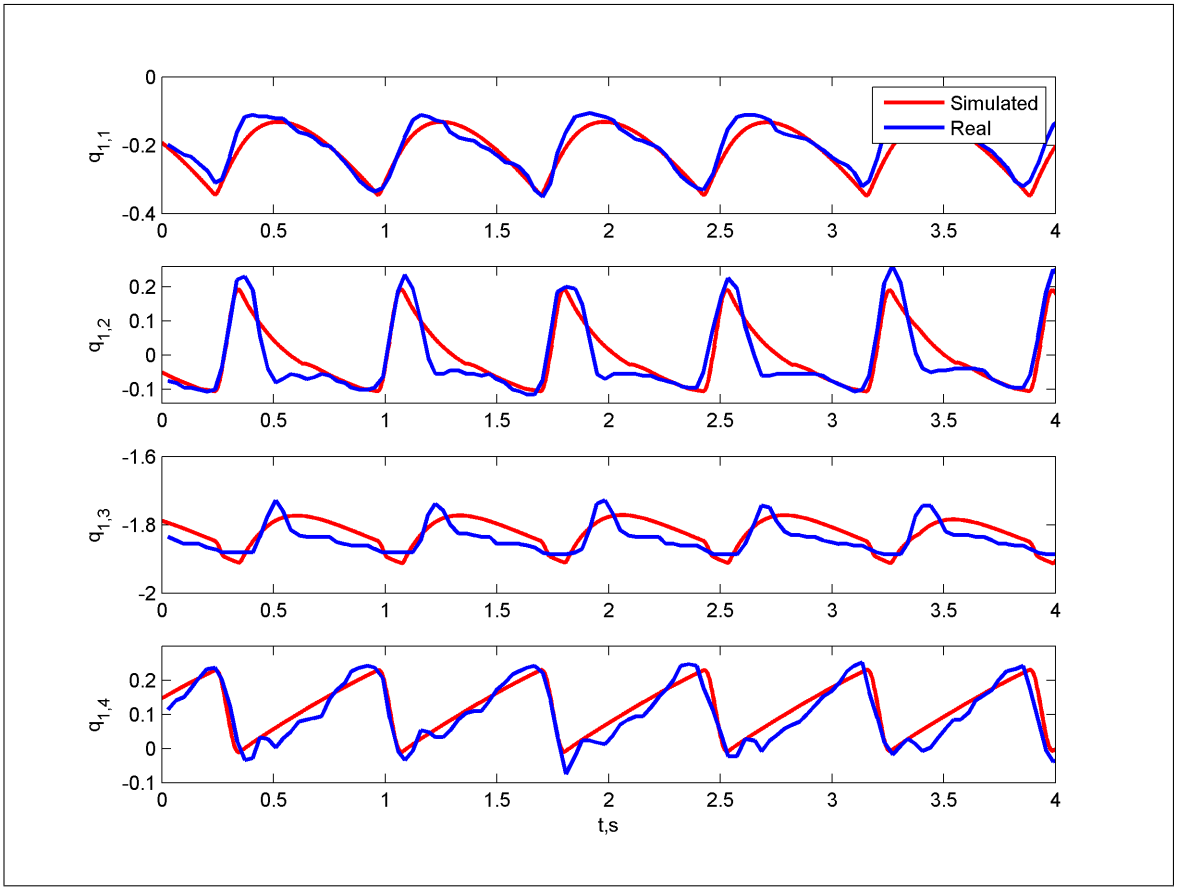

Figure 16: Joint position comparison between the simulated and real quadruped while performing a gait which achieves a forward velocity of $60\frac{mm}{s}$

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 (4.21). 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 *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 return are corrupted by zero-mean Gaussian random noise with variances close to respective parameters which describe the physical LIDAR unit.