A Python Framework for Legged Systems

Shruti Misra
Department of Electrical Engineering
University of Washington
Seattle, WA
shrm145@uw.edu

***Abstract*—This report provides a Python based framework to simulate gaits of different legged systems. The framework builds upon the MATLAB framework as provided in [1]. It provides examples of two different legged systems, namely the Passive Dynamic Biped and the Bounding Quadruped. Both the systems are treated as hybrid dynamical systems and are thus modeled as such. Therefore, building the Python framework provided a good insight into understanding and modeling complex hybrid dynamical systems from a computational point of view.**

Keywords—Hybrid dynamical systems; Flow Map; Jump Map; Jump Set;

I. INTRODUCTION

Legged systems usually fall under the category of hybrid dynamical systems. Hybrid dynamical systems can be thought of as a combination of properties of continuous and discrete systems. In case of legged locomotion, when a foot/feet collide(s) with or leave(s) the ground, the continuous dynamics of the legged body are disrupted and need alteration. The instances when these disruptions occur are known as events. The "jump map" dictates how the system will behave when an event occurs. The "jump set" (also known as the "guard set") of the system determines when an event has occurred. The occurrence of an event is commonly characterized by a zero crossing of some state or function that is being monitored during simulation. The "flow map" directs the continuous dynamics of the system.

This paper builds upon the MATLAB framework for legged systems as provided in [1] and attempts to convert it to Python. The advantages of Python over MATLAB are two-fold. Firstly, Python compilers are available on most operating systems, free of charge, unlike MATLAB. Secondly, Python code can run on microcontrollers such as the Beaglebone, and thus can directly be used to run on real robotic systems. Two examples of legged systems are provided in this paper; the passive dynamic biped and the bounding quadruped. The former system as rightly named is passive and does not involve any actuation. The bounding quadruped on the other hand is modeled as an activated system, where the activation function was obtained from [1] and [2]. The paper first discusses some basic theory regarding hybrid systems. It then delves into the theoretical details of each example. The paper then examines the results obtained by the Python model and compares it with the results obtained from the MATLAB model. Finally, conclusions are drawn about the process of porting the MATLAB code over to Python. Therefore, the product of this paper can be regarded as a tool to aid in further research into legged systems with a Python based framework.

II. THEORY

A. Hybrid Dynamical Systems

As previously explained, hybrid dynamical systems combine aspects of continuous and discrete systems. When the continuous dynamics of the system are disrupted by some system specific occurrence, the system undergoes a discrete transition. These "disruptive" occurrences are known as events. Events can either be forced or switched. In a switched system, the discrete transition is allowed to occur. In contrast, a forced system is where the system is compelled to undergo a discrete transition. A forced system does not have control over when it can undergo a discrete transition. There are three major sections of a hybrid dynamical system and they are outlined below

Flow Map

The flow map dictates the continuous dynamics of the system and can be represented as,

$$\dot{x}_{cont} = f(x, p) \qquad (1.1)$$

In the above equation, x represents a vector of all the states of the system and p represents the system parameters. In the case of activated systems, an addition u term is added to denote the inputs that affect the continuous dynamics. The flow map usually contains a set of differential equations, which in case of legged locomotion applications, comprise of equations of motion that describe how the legs and the robot body move.

Jump Set

The jump set determines when an event occurs and thus determines when a discrete transition occurs. The jump set of the system can be defined by directional zero crossings of a state or an event function and can be written as,

$$e(x, p) = 0 \; with \; \dot{e} > 0 \quad (1.2)$$

In Equation (1.2), x represents a vector of the current states of the system and p refers to the system parameters, which maybe adjustable. In the case of legged locomotion, the jump set detects when one or more feet strike the ground.

Jump Map

The jump map defines how a discrete transition occurs, given the current state of the system. It provides a response to an event occurrence and can be written as,

$$x^+ = g(x^-, p) \qquad (1.3)$$

In the above equations, x^+ and x^- denote the states of the system right after and right before the event respectively. For legged systems, equation (1.3) can be an implementation of impulse equations for mechanical collision.

In the framework presented by Remy,

$$x = [y, z]^T \qquad (1.4)$$

The above equation represents the fact that the state variables are divided into two distinct entities; continuous states (y) and discrete states (z). The distinction is justified for performance purposes but is not necessary. In the basic implementation of the system, the x vector consists of the generalized coordinates $\mathbf{q}$ and generalized speeds $\dot{\mathbf{q}}$. However, x can be extended to encompass more states that can keep track of other variables, such as energy consumption or time passed since the last event.

B. Multi-Body Dynamics

The dynamical structure of legged systems changes when the foot or feet strike(s) the ground. Therefore, there is a need for a formulation of the dynamics such that they consist of unilateral constraints. Such a multi body system can be represented by the following equation of motion,

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{f} + \mathbf{J}^{\mathbf{T}}(\mathbf{q})\boldsymbol{\lambda} \qquad (1.5)$$

Where,

M = Mass matrix of the system

q, $\dot{q}$, $\ddot{q}$ = generalized coordinates, speeds and accelerations respectively

h = Differential force vector (which includes gravitational, Coriolis and centrifugal forces)

f = Generalized forces in joints, such as friction and actuation

J = Contact Jacobian, obtained by the partial differentiation of the contact vector **r** and maps a vector of contact forces λ to the generalized coordinate space.

It is assumed that feet on the ground can only perform a pure rolling motion; no slipping or sliding motion is modeled. Therefore, for contact distance vector **r,**

$$\dot{\mathbf{r}} = \mathbf{J}\dot{\mathbf{q}} = 0, \text{where } J = \frac{\partial \mathbf{r}}{\partial \mathbf{q}} \qquad (1.6)$$

$$\ddot{\mathbf{r}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}} = 0 \qquad (1.7)$$

Equation (1.5) can be solved for $\ddot{\mathbf{q}}$ and substituted in equation (1.7) to solve for the contact forces λ. The contact forces can then be used to compute the generalized accelerations ($\ddot{\mathbf{q}}$) in the flow map. The contact forces are also used in the jump set to determine when a foot leaves the ground and thus opens a contact. A contact opens, when its corresponding normal force becomes negative. In contrast, the closing of a contact occurs when the foot touches the ground. The jump set detects this when the vertical position of a particular contact (feet) is about to become negative.

When a foot makes contact with the ground, instantaneous changes in velocities occur which are modeled by a perfectly inelastic collision. Since the collision occurs over a very short time span, the effect of **h** and **f** can be ignored. Only the impulsive forces and difference in velocities must be accounted for. The external impulsive forces (Λ) are computed by integrating the equations of motion over the span of the collision.

$$\int_{\{t0\}} (\mathbf{M}\ddot{\mathbf{q}} - \mathbf{h} - \mathbf{f} - \mathbf{J}^{\mathbf{T}}\boldsymbol{\lambda})dt = \mathbf{M}(\dot{\mathbf{q}}^+, \dot{\mathbf{q}}^-) - \mathbf{J}^{\mathbf{T}}\boldsymbol{\Lambda} = 0 \qquad (1.8a)$$

Assuming a perfectly inelastic collision, the points of the feet that are considered a part of the collision, instantly come to rest. Moreover, for contacts that open simultaneously, the contact points must leave the ground right after the collision. The above ideas are expressed in the following description of the collision [2],

$$\dot{\mathbf{r}}^+ - \dot{\mathbf{r}}^- = \mathbf{J}.(\dot{\mathbf{q}}^+ - \dot{\mathbf{q}}^-) = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathbf{T}}.\boldsymbol{\Lambda} \qquad (1.8b)$$

$$\dot{r}^{y+} \geq 0, \Lambda^y \geq 0, \dot{r}^{y+}.\Lambda^y = 0$$

The above equation can be solved for post impact speeds $\dot{\mathbf{q}}^+$.

III. Models

As mentioned previously, examples of two separate legged systems have been ported over from MATLAB to Python. This section summarizes the dynamics of each example.

A. Passive Dynamic Biped

Passive Dynamic Biped is a category of legged systems that can walk down a mild slope without using any actuation.

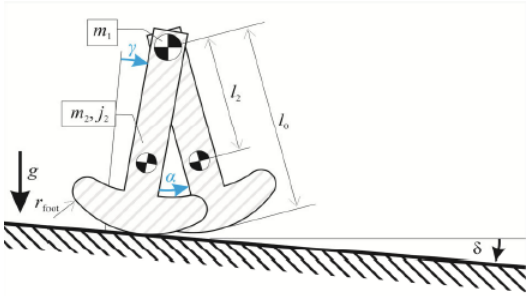

Figure 1: A passive dynamic walker [1]

As seen in Figure 1, the system consists of three portions: a main body with mass m_1, and two legs with distributed mass (m_2, j_2). The legs are of length l_0 and end in circular feet with radius r_{foot}. The position of the center of gravity of the legs denoted by l_2 can be varied. The slope that the biped walks down is given by δ .It is assumed that one foot is rolling on the ground at all times while the other one is swinging. The leg belonging to the feet that is rolling is known as the stance leg and the leg that is swinging is known as the swing leg. The dynamics of the system is disrupted when the swing leg strikes the ground, that is, when the role of the stance and the swing leg is switched.

The system states are given as follows,

$$x = [\gamma, \dot{\gamma}, \alpha, \dot{\alpha}]^T \qquad (1.9)$$

Where,

α = Inter leg angle between both the legs

γ = Angle between the ground and the stance leg

The system does not have any discrete states. Tables 1 and 2 provide the initial conditions and the system parameters as used in the MATLAB framework

Table 1: Initial conditions for the Passive Dynamic Biped as given in the MATLAB framework

States	Initial Values
γ	0.3 radians
$\dot{\gamma}$	-0.5 radians
α	-0.6 radians
$\dot{\alpha}$	0.5 radians

Table 2: System parameters as given in the MATLAB framework. The units are normalized relative to the total mass and leg length

System Parameters	Value
Leg length (l_0)	1
Mass of main body (m_1)	0.20
Mass of legs (m_2)	0.40
Distance between hip joint and CoG of the legs (along the legs) (l_2x)	0.5
Distance between hip joint and CoG of the legs (perpendicular to legs) (l_2y)	0
Foot radius (r_{foot})	0.5
Inertia of legs (j_2)	0.002

Flow Map

The flow map of the system is given by the following set of equations

$$\dot{\gamma} = x_2 \qquad (2.0)$$

$$\dot{\alpha} = x_4 \qquad (2.1)$$

From Equation (1.5), a simplified equation of motion for the continuous part of this system can be derived and is given by,

$$\mathbf{M\ddot{q}} = \mathbf{h} \quad (2.2)$$

The above equation can be solved for $\ddot{\mathbf{q}}$ because the Mass Matrix and the force matrix **h** are known. Therefore, the rest of the equations for the flow map are given by,

$$\ddot{\gamma} = \ddot{\mathbf{q}}_1 \qquad (2.3)$$

$$\ddot{\alpha} = \ddot{\mathbf{q}}_2 \qquad (2.4)$$

Thus, equations (2.0), (2.1), (2.3) and (2.4) dictate the continuous dynamics of the system.

Jump Set

According to the MATLAB framework, the jump set is defined when the contact height of the swing leg is approaching zero and when the swing leg is in front of the stance leg, this is represented by the following equation

$$contact\ height\ (h) = 0, \text{where } x_3 > 0 \qquad (2.5)$$

The contact height for the system is given by,

$$h = -(\cos(x_3 + x_3) - \cos(x_3))(l_0 - r_{foot}) \quad (2.6)$$

Jump Map

The new angles right after the discrete transitions can be directly computed because it just reflects the switching of the stance leg and the swing leg. The following equations are used for this computation

$$\gamma^+ = \alpha^- + \gamma^- \quad (2.7)$$

$$\alpha^+ = -\alpha^- \qquad (2.8)$$

Computing the post collision velocities is slightly harder, as a full plastic collision needs to be calculated. Therefore, the first step is to calculate the hip velocities before the collision and switch the roles of the stance and swing legs. Next, a full plastic collision is computed in accordance with equation (1.8). Thus, a matrix with the contact velocities is obtained and employed [2].

B. Bounding Quadruped

The bounding quadruped example is a legged system with 4 legs and a body. The legs in this model are actuated by series elastic actuators. The actuators are excited periodically with parametric oscillations. The figure below provides a 2D version of the quadruped model.

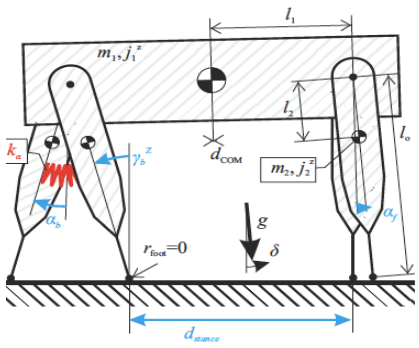

Figure 2: Planar version of the bounding quadruped model

The main body of the quadruped measures $2l_1$ from hip to shoulder and has an inertia of j_1^z about the medio-lateral (M-L) axis, of j_1^y about the dorso-ventral (D-V) axis, and of j_1^x about the antero-posterior (A-P) axis. Its center of mass is initially in the middle of the main body, but can be displaced along the antero-posterior axis by a distance of d_{COM}. The legs (m_2, $j_2^{x,y,z}$) are connected by single degree of freedom rotational joints at the hip and shoulder, where each pair of legs shares a common rotational axis normal to the sagittal plane of the model. The horizontal spacing between the legs is given by d_{lat}. Their center of mass is situated l_2 below the joints [2]. Torsional springs (with stiffness $k\,\alpha$) connect the stance and swing legs of each leg pair. In this model, two feet are in stance at all times.

The system states are divided into continuous states y and discrete states z. Tables 3 and 4 provide the continuous and discrete states, along with their initial values and description. Figure 3 is a summary of the system parameters, their values and description.

Table 3: Continuous states of the system

States	Description	Values
x	Horizontal position of main body	0.00
dx	Horizontal velocity of body	0.70
y	Vertical position of main body	1.15
dy	Vertical velocity of body	0.00
phi	Pitch of main body	0.01
dphi	Angular pitch velocity	0.15
alphaF	Angle of the front leg wrt the main body	-0.32
dalpha F	Angular velocity of above	0.64
lF	Length of front leg	1.07
dlF	Velocity of above	-0.57
alphaB	Angle of the back leg wrt the main body	0.01
dalpha B	Angular velocity of above	0.81
lB	Length of back leg	0.97
dlB	Velocity of above	0.01
time	Time passed since start of step	0.00

Table 4: Discrete States of the system

States	Description	Initial Value
phaseF	Current phase of front leg	2
phaseB	Current phase of back leg	2
COT	Cost of transportation	0

Param.	Value	Unit	Description
m_o	1	$[\cdot]$	Total mass
l_o	1	$[\cdot]$	Leg length
g	1	$[\cdot]$	Gravitational constant
m_1	0.8	$[m_o]$	Main body mass
l_1	0.75	$[l_o]$	Main body length
$j_1^{z,y}$	0.18	$[m_o l_o^2]$	Main body inertia about the M-L and D-V axes
j_1^x	0.072	$[m_o l_o^2]$	Main body inertia about the A-P axis
m_2	0.05	$[m_o]$	Leg mass
$j_2^{x,z}$	0.0022	$[m_o l_o^2]$	Leg inertia short axes
j_2^y	0.0005	$[m_o l_o^2]$	Leg inertia long axis
l_2	1/3	$[l_o]$	Leg COM position
d_{lat}	0	$[l_o]$	Lateral leg spacing
d_{COM}	0	$[l_o]$	Offset of the COM
δ	1	$[°]$	Ground inclination
k_α	0	$[m_o g l_o/\mathrm{rad}]$	Hip/shoulder spring stiffness

Figure 3: System Parameters of the bounding quadruped

Flow Map

As can be seen in Table 3, almost all states have a corresponding derivative as a state variable. Therefore, the first and easiest step is to set the "non derivative" states to their derivative from the previous time step. The complicated part of the flow map is to compute the "derivative states" for the system (such as dx, dy, dphi etc.). This step depends on the current phase of the system, which is given by the discrete states. There are four possible phases of the legged system. The phase number is computed using the following equation,

$$phase = phaseF + 2 * phaseB \qquad (2.9)$$

Table 5: Possible phases of the four legged system

Phase	phaseF	phaseB
3	1	1
4	2	1
5	1	2
6	2	2

Table 5 provides an overview of all the phases that the system can hold and the corresponding values of the relevant discrete states. The value of '1' or '2' of a discrete state denotes where the leg is in stance or flight phase respectively. Therefore, if phaseB is 1 and phaseF is 2, this represents that the back leg is in stance and the front leg is in flight respectively. Therefore, the contact Jacobian is computed for the leg that is in contact with the ground, which in turn allows

us to compute the contact forces for that phase. For example, if the system is in phase 3, then the contact Jacobian and contact force matrix is a combination of both the front and the back leg, because one or both of the front and back legs are in contact, in that phase. In phase 6 however, none of the legs are in contact and thus, the contact forces in this case will be the 0 vector. Moreover, differential forces such as the gravitational force, Coriolis force and actuator forces are computed as well but are phase independent. Therefore, the equation of motion for the continuous part of the system is given by,

$$\mathbf{M\ddot{q}} = \mathbf{h} + \mathbf{J^T\lambda} \quad (3.0)$$

Thus, equation (3.0), allows the computation of "derivative states", or states that hold the derivative of other states.

Jump Set

The jump set as described in the MATLAB framework is designed to detect 5 different events. The first event detects the touchdown of the front leg and the second event detects the liftoff of the back leg.. Similarly, the third and fourth events detect the touchdown and liftoff of the front and back leg respectively. The fifth and the final event detects the stopping point for an active system.

The touchdown events are detected by checking if the front or the back foot goes below the ground during flight. This is similar to the jump set function of the Passive Dynamic Biped. The liftoff events are slightly complicated as they involve computing the contact forces as done in the flow map and then checking if the contact forces are going negative. This is because during liftoff, the foot is leaving the ground and therefore, the contact between the ground and the foot is decreasing, which results in decreasing forces between them. The fifth event seems important under the MATLAB framework, but under the Python implementation it does not really have any special use. The only observation that is made about the fifth event is that, the system will not undergo any changes if the fifth event occurs. So, in a way it represents the terminal event. The jump set outputs an event vector, the size of the number of events. Events that are not triggered have a value of -1 in their corresponding index in the event vector. In contrast, an event that has been triggered has a negative value in their corresponding index. Only one event can be triggered at any given time.

Jump Map

The jump map results in the discrete transition of the system states based on which event has been triggered in the jump set. In the case of liftoff events, the velocities and the other state variables remain unchanged; the only states that need to be altered are the discrete states, to reflect which feet are in flight mode. However, for touch down events, the process is similar to computing the jump map for the Passive Dynamic Biped. The contact forces and hip velocities for the relevant legs in contact need to be computed and thus the velocities after collision are calculated in accordance with Equation (1.8). Moreover, in cases of touchdown, occurrence of double contact needs to be checked for. That is, if the front leg has touched down, a check needs to be made by observing the discrete state phaseB to determine if the back leg is also in stance. If so, then the contact force matrix and subsequently, the collision velocities will change based on the fact that there is a double collision.

Excitation Function

The Bounding quadruped is an activated system powered by series elastic actuators. The system consists of four actuators, the frontal and back rotational actuators and the frontal and back linear actuators. The MATLAB framework defines excitation parameters, which consists of sine and cosine terms for the front and back leg rotation and extension. Moreover, the excitation parameter vector also comprises of the stride frequency as a parameter, which is set to 0.5. This value is normalized with respect to gravity, total mass and uncompressed leg length of the system. The excitation function is just a Fourier series input to the actuators based on the excitation parameters. The general equation for the Fourier input is given below

$$u_{actuator} = u_{actuator} + \sin_{actuator}(A)\sin(n\phi) + \cos_{actuator}(A)\cos(n\phi) \quad (3.1)$$

$$\phi = 2\pi * (stride\ frequnecy) \quad (3.2)$$

Where, 'A' is the amplitude of the corresponding sine and cosine term and 'n' is the number of oscillations (or number of periods)

IV. Results

The main goal of this project was to port the MATLAB framework provided in [1] for each example system, to a Python implementation. Testing was conducted by comparing the results obtained from the MATLAB code with those obtained in Python. The Python implementation for each example, was built on the dynamics discussed in Section III. Certain changes had to be made while porting the code over and are discussed in Section V. This section largely verifies the results obtained in Python by comparing them with results obtained in the MATLAB framework designed in [1]. Therefore, the comparison provides evidence of the proper functioning of the Python framework.

A. Passive Dynamic Biped

Figures 4 and 5 illustrate the flow map of the Passive Dynamic Biped system obtained from MATLAB and Python respectively.

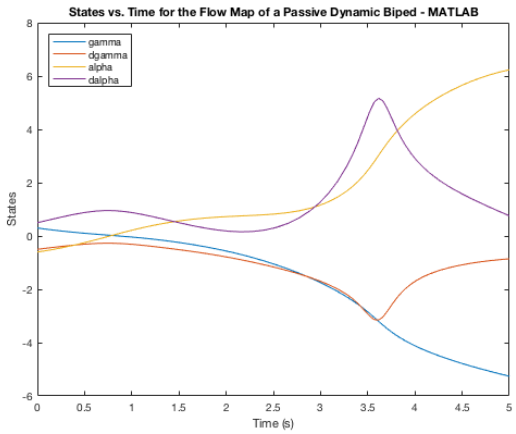

Figure 4: Flow Map obtained from the MATLAB implementation of the Passive Dynamic Biped system

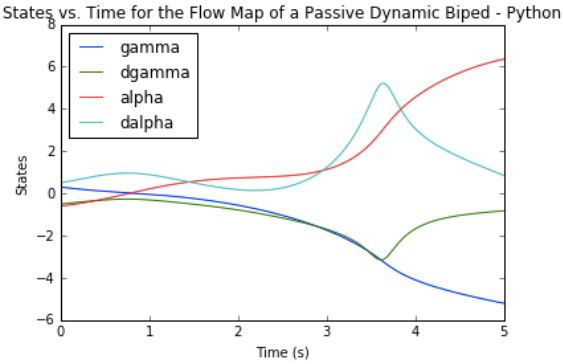

Figure 5: Flow Map obtained from the Python implementation of the Passive Dynamic Biped system

The above figures are nearly identical thus proving that the flow map implementation in Python is functional. Since this system does not have any discrete states, the jump map for the system was not plotted.

Figures 6 and 7 depict the integrated Passive Dynamic Biped system in MATLAB and Python respectively. In this case, all the aspects of the system such as the flow map, jump map, jump set etc. are simulated together and thus provide an overall illustration of how the system works.

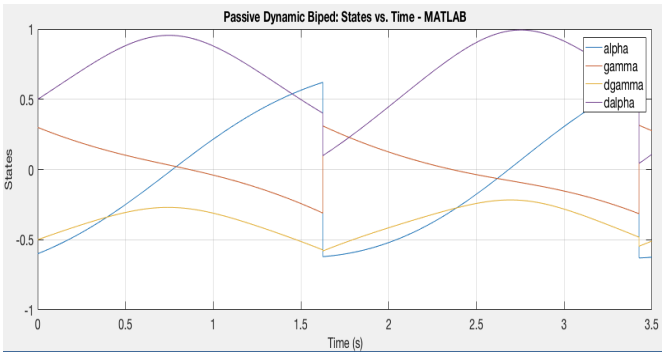

Figure 6: Passive Dynamic Biped system in MATLAB

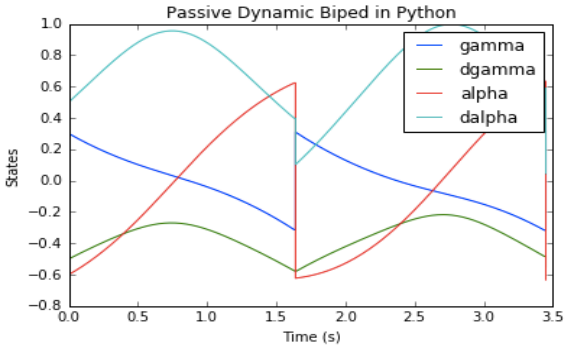

Figure 7: Passive Dynamic Biped system in Python

Again, figures obtained from the MATLAB and Python framework are nearly identical with some slight differences in the y-axis scale due to different simulation functions. Therefore, it can be inferred that the Python implementation has been successfully implemented for the Passive Dynamic Biped.

B. Bounding Quadruped

The flow map of the system as obtained from MATLAB and Python is presented in Figures 8 and 9 respectively. Due to the presence of a lot of state variables is it a little difficult to follow each state in the flow map. However, upon doing so, it can be observed that the flow maps from MATLAB and Python are almost the same except for some minor scaling differences. Similarly, figures 10 and 11 illustrate the jump maps of the system derived from MATLAB and Python respectively. In this case too, both the maps are identical except for slight differences in scaling. Therefore, it can be concluded that the flow and jump map for the system have been successfully implemented in Python.

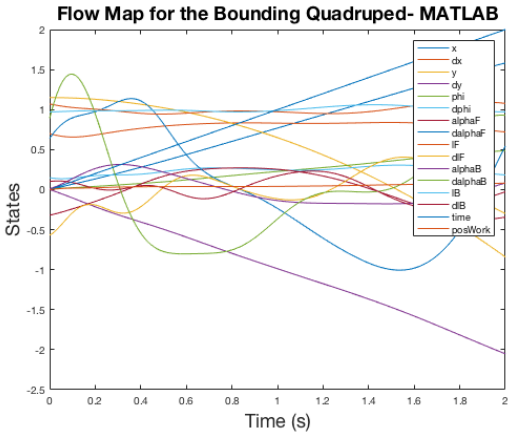

Figure 8: Flow Map obtained from the MATLAB implementation of the Bounding Quadruped

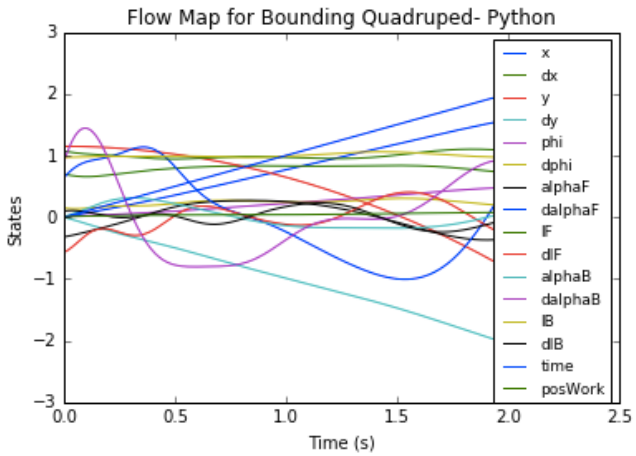

Figure 9: Flow Map obtained from the Python implementation of the Bounding Quadruped

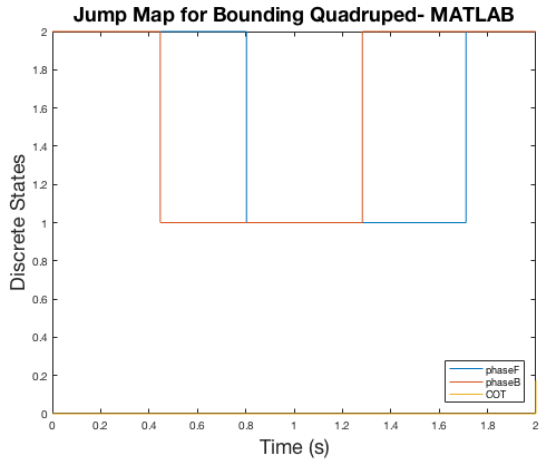

Figure 10: Flow Map obtained from the Python implementation of the Bounding Quadruped

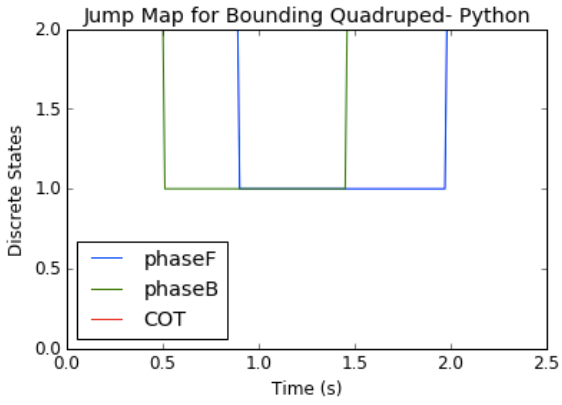

Figure 11: Flow Map obtained from the Python implementation of the Bounding Quadruped

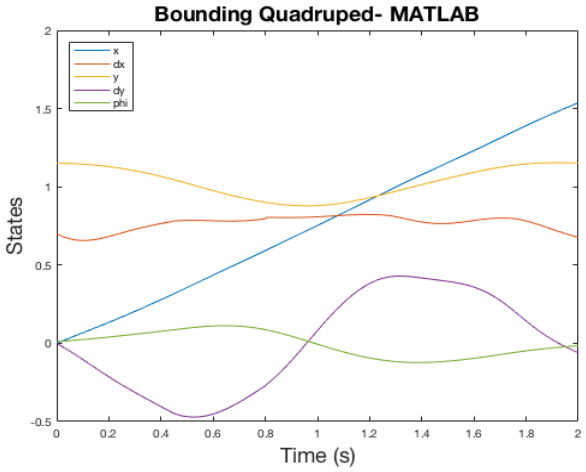

Figure 12a: First five states of the Bounding Quadruped system in MATLAB

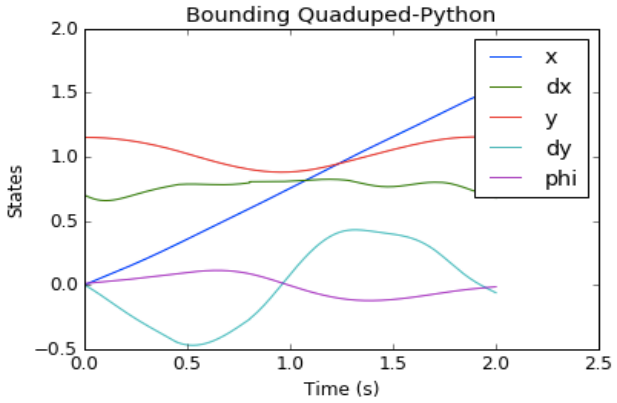

Figure 12b: First five states of the Bounding Quadruped system in Python

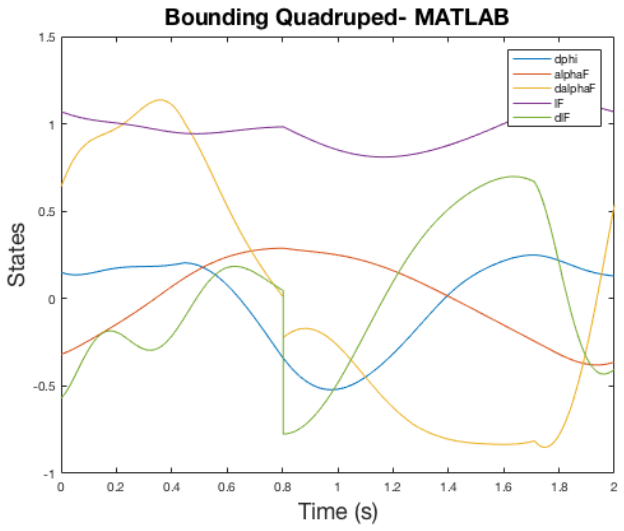

Figure 13a:Next five states of the Bounding Quadruped system in MATLAB

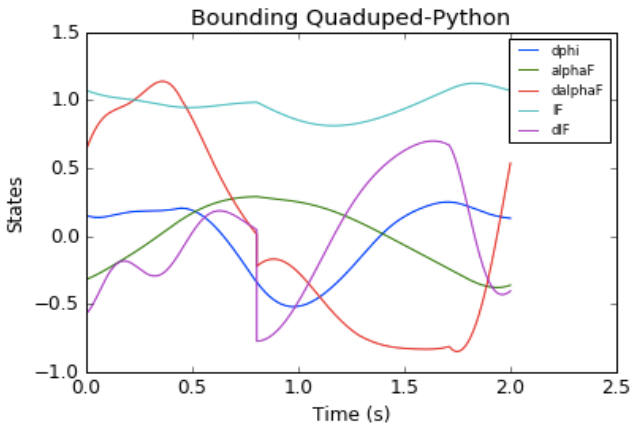

Figure 13b:Next five states of the Bounding Quadruped system in Python

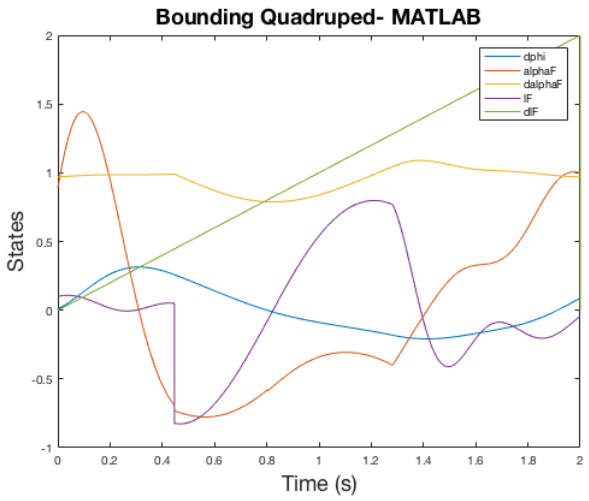

Figure 14a:Last five states of the Bounding Quadruped system in MATLAB

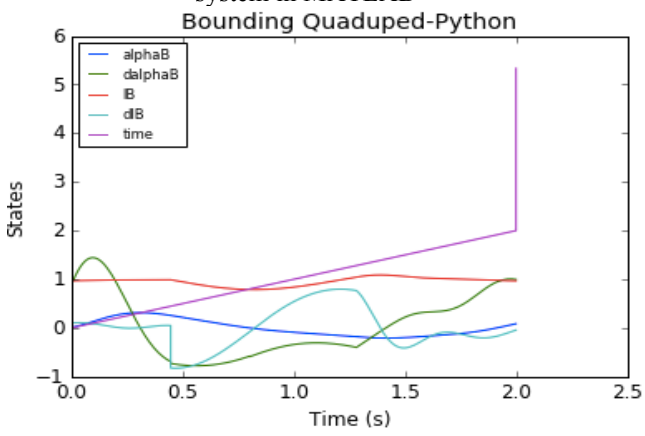

Figure 14b:Last five states of the Bounding Quadruped system in Python

Figures 12, 13 and 14 compare the states of the integrated Bounding Quadruped system in MATLAB and Python respectively. Since the system consists of a total of 15 states, they are divided into 3 sets of 5 states per figure for clearer analysis. Again, it can be observed that the graphs obtained from the MATLAB framework are identical to those obtained from the Python implementation. In figures 14a and 14b, the scale is a little different, this is because the system reaches the stopping point in both cases, and thus the simulated time increases to reflect the step number, which is step or event 5. Thus, upon inspection of the above figures, it can be concluded that the Python implementation for the Bounding Quadruped system is successful.

V. Discussion

While the Python framework is quite identical to the MATLAB implementation, there are some differences in the structure of the code and how the systems were simulated. The MATLAB implementation is somewhat more sophisticated as it provides options to include excitation functions to all the systems, conducts optimization and finds periodic gaits. The Python framework is not as flexible, it does not provide an easy way to add an excitation function to passive systems and does not conduct any optimization or periodic gait search. Moreover, it does not implement a high-tech file structure, indexing system or working animations. The Python implementation is quite bare bones and just consists of a class for each model. All the functions are encompassed in this class, with some additional dependencies, which contain the computation for the mass matrix, contact forces etc. This can be viewed as an advantage, because all the code is in one place and thus there is a clear flow of how the variables are manipulated. However, this also makes the implementation more cumbersome.

Moreover, the MATLAB implementation “auto-generates” most of the functions that deal with computing Jacobians, mass matrices, contact dynamics and differential forces. This is done with the help of the Symbolic Toolbox in MATLAB, where the equations of motions are described symbolically and the auto-generated functions compute the symbolic equations when provided with numerical values. No such method has been implemented in Python as there seemed to be no way to “auto-generate” functions explicitly. Work arounds for this, in Python were not explored extensively. Therefore, most of these computation heavy, “auto-generated” functions were implemented manually in Python. This is a potential problem should new models be included in the Python framework, because there is no explicit function within the code base to compute equations of motions and thus obtain vital entities such as the mass matrix, contact Jacobians etc. Therefore, the Python implementation is quite basic and inflexible, however it can be improved upon in the future to make it more robust and adaptable.

Another main difference between the Python and MATLAB code is how the systems are simulated. MATLAB uses ODE45 with an inbuilt, sophisticated event detection function, that can detect zero crossings quite accurately. The Python framework uses Forward Euler and fourth order Runge-Kutta to simulate the continuous dynamics. The event

detection portion of the simulation just checks the jump set values for when they are negative, to activate and apply the reset map. This has worked quite well because the systems employ directional zero-crossings. This might not be as efficient for bidirectional zero crossings (if ever necessary), as detecting when the function is nearing zero is difficult in Python especially if it needs to be checked from both directions. There are some libraries such as PyDSTool that implement event detection, however this might involve redoing a large portion of the current framework to work with the said library.

VI. Conclusion

The aim of this paper was to understand the MATLAB framework for legged systems presented in [1] and port it over to Python. This required understanding the theory behind legged systems so that it can be applied to simulate the system computationally. Two examples of legged systems were studied, namely the Passive Dynamic Biped and the Bounding Quadruped. Both these examples were modeled as hybrid dynamical systems. The paper recapitulated some basic yet context specific hybrid dynamics theory. It also delved into the system specific theory for both the examples. Results from the reference MATLAB framework were compared to the newly built Python implementation. It was found that the Python results mimicked the MATLAB results quite closely, thus providing evidence to the fact that the example systems were successfully ported to Python. The two frameworks are different from each other in terms of their structure and simulation technique. The Python code is quite basic and a lot less sophisticated in structure and simulation technique. However, it can be refined to be more robust. It provides a good foundation to build upon for more complex functionalities such as optimization, periodic gait search and even including other legged system models.

References

[1] C. D. Remy, K. W. Buffinton, and R. Siegwart. A matlab framework forefficient gait creation. In International Conference on Intelligent Robots and Systems, IROS , page (accepted for publication), 2011b.

[2] C. D. Remy,. Optimal Exploitation of natural dynamics in legged locomotion. ETH (2011).