The field of autonomous robotics endeavors to engineer systems capable of transforming sensory inputs into motors outputs toward the achievement of goals. Ideally, the system would form novel motor behaviors appropriate to its environment of operation while concurrently learning the conditions of satisfaction for goals set by the roboticist. Unfortunately, the space of sensory inputs and motor outputs is often vast, posing a ghastly combinatorial selection problem, and goals are often overly restrictive or vague. For example, imagine a roboticist builds a system with visual, tactile, and proprioceptive sensors as well as sufficient actuators to control is limbs. Perhaps she wishes for it to approach red objects within the environment. Does the robot know how to produce coordinated movements? Does it know how to approach an object? For that matter, how does it know what an “object” is? and what is meant by “red”? In this paper we focus upon the use of chaos control as a means for generating coordinated movements in robotic systems. In particular, we examine the work of Steingrube et al. 2010, wherein chaos control of a single neural control circuit was used to produce multiple motor outputs conditional upon a variety of sensory inputs. We simulate some of their results and critique their autonomous system according to the amount of information programmed into it a priori. We then show how a recent advance in computer science, self-localizing and mapping (SLAM)systems, may vastly increase the memory systems of autonomous robots in general, and those utilizing chaos control in particular. We propose a generalization of SLAM that, when coupled with principles from optimal control theory, could greatly expand the functionality and independence of autonomous systems. We conclude by articulating some outstanding questions in the field while arguing that engineering systems with the ability to answer the kinds of questions enumerated above is as much a phenomenological and conceptual problem as it is a technical one.

Chaos Control for motor pattern generation

Chaos control offers a promising direction for the design of a single circuit that allows a system to explore and select trajectories through complex motors spaces. The attractive feature of chaotic attractors that suggests their use is their having embedded within them an infinite number of unstable periodic orbits (Ott et al. 1990). This means that a single chaos generator could generate a large repertoire of periodic behaviors (Steingrube et al. 2010) e.g. walking, running, turning, striking, to name a few. It is worth emphasizing that gaining control over multiple outputs generated by a single substrate is not a trivial task. The pioneers of autonomous robotics were aware of the difficulty
in designing systems capable of “cumulating adaptations” when this always seemed to require dividing its mechanisms into “non-overlapping sets” (W. Ross Ashby 1960). The central challenge is enabling the system to learn multiple associations between sensory inputs and motor outputs without these associations interfering with each other. The conventional approach is to “hard wire” a specific circuit for each behavioral response at the expense of adaptation. In contrast, a chaotic generator might be multipurpose and flexible.

We start by examining the control law employed by Steingrube et al. 2010. They rely heavily upon the linearization techniques developed by Ott et al. 1990 and Schmelcher and Diakonos 1997. Our interest is in how this control scheme both detects and stabilizes the unstable periodic orbits of a single chaotic system.

Let’s begin by recapitulating the main equations used by Steingrube et al. They defined a 2-neural module system (Pasemann 2002) as

\[ x_i(t + 1) = \sigma(\theta_i + \sum_{j=1}^{2} w_{ij}x_j(t) + c_i^{(p)}(t)) \text{ for } i \in 1, 2 \]

For a given period \( p \), the control signal

\[ c_i^{(p)} = \mu^{(p)}(t) \sum_{j=1}^{2} w_{ij}\Delta_j(t) \]

depends on the differences between states, i.e. error signal,

\[ \Delta_j(t) = x_j(t) - x_j(t - p) \]

separated by one period \( p \) and was applied every \( p+1 \) time steps (\( \Delta_j = 0 \) and thus \( c_i^{(p)} = 0 \) at all other times) such that each point of a periodic orbit is controlled sequentially. (The motivation for this sequential updating is found in Schmelcher and Diakonos 1997 and will be discussed below). The control strength \( \mu^{(p)} \) adapts according to

\[ \mu^{(p)}(t + 1) = \mu^{(p)}(t) + \lambda \frac{\Delta_1^2(t) + \Delta_2^2(t)}{p} \]  

It is important to note that \( \mu(t_{\text{initial}}) = -1 \) whenever \( p \) changes and \(-1 \leq \mu^{(p)} \leq 0 \). Parameter values, weights \( w_{ij} \) and \( \theta_i \), were chosen such that the system was chaotic in the absence of control. The chaotic parameter regime for this system is detailed elsewhere (Pasemann 2002).
Where does this control law come from? Firstly, expanding (1) upon application of the control law and rearranging terms

\[
\begin{align*}
    x_1(t + 1) &= a \left( \theta_1 + w_{11}[x_1(t) + \mu^{(p)} \Delta_1(t)] + w_{12}[x_2(t) + \mu^{(p)} \Delta_2(t)] \right) \\
    x_2(t + 1) &= a \left( \theta_2 + w_{21}[x_1(t) + \mu^{(p)} \Delta_1(t)] \right)
\end{align*}
\]

(5) (6)

exposes the linearization of the system as

\[x_i(t + 1) \approx x_i(t) + \mu^{(p)}(x_i(t) - x_i(t - p)) \text{ for } i \in 1, 2 \]

(7)

Note that by substituting in \(\mu(t_{\text{initial}})\) for \(\mu^{(p)}\) in (7) and distributing the negative sign, (7) becomes exactly a first order taylor expansion about \(x_i(t)\) and the bounds upon \(\mu\) become \(0 \leq \mu \leq 1\). This tells us that we are searching for a fixed point of period \(p\) somewhere on the line between \(x_i(t)\) and \(x_i(t - p)\). Now we must ascertain how this linearization stabilizes periodic orbits.

Linearization techniques are central to chaos control. The approach of Ott et al. 1990 requires finding an update for the control parameter \(p\) (in their case this is one of the system parameters) such that the matrix \(M\) in the linearization,

\[\xi_{n+1} \approx \xi_F(p) + M \cdot [\xi_n - \xi_F(p)]\]

(8)

drives the chaotic system along a stable eigenvector to a fixed point. This necessitates estimating the unstable and stable eigenvalues and eigenvectors of the chaotic system about some desired fixed point. They only mention the significant challenge of estimating these scalars and vectors in passing, which belies the fact that this is a non-trivial task (Ghil et al. 2002). Alternatively, Schmelcher and Diakonos 1997 demonstrated that it is possible to stabilize the fully chaotic dynamical system

\[U : \vec{r}_{i+1} = \vec{f}(\vec{r}_i)\]

(9)

by finding a linear transformation \(L_k : U \rightarrow S_k\) with \(S_k\) defined as

\[S_k : \vec{r}_{i+1} = \vec{r}_i + \Lambda_k(\vec{f}(\vec{r}_i) - \vec{r}_i)\]

(10)

such that the fixed points of \(S_k\) match those of \(U\) and are stable. They search among the set of orthogonal matrices \(C_k\) with elements \(C_{ij} \in 0, \pm 1\) to find some \(C_k\) such that \((\Lambda_k)_{ij} = (\lambda C_k)_{ij}, 0 \leq \lambda \leq 1\) has all negative eigenvalues, and is therefore stable. The number of possible \(N\)-dimensional orthogonal by this construction is \(N!2^N\). Although
exponential in the number of dimensions, each \( C_k \) may be tested sequentially until a member of the subset of stabilizing \( C_k \) is found.

For the chaotic system of Steingrube et al. 2010 the stabilizing matrix is simply the unit matrix:

\[
C_k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\]

Hence, the stabilizing matrix \( C_k \) is implicit in their control law and \( \mu^{(p)} \rightarrow \lambda \). But how did they detect the unstable periodic orbits in the first place? Once again, they relied upon a modification of the methods of Schmelcher and Diakonos. With \( C_k \) in hand one simply picks an initial point from the chaotic time-series and attempts to stabilize it. If this leads to a stable fixed point, the sequential updates do nothing because the system is deterministic. If this initial point does not lead to a stable fixed point, the sequential updates searches through the chaotic time-series for an initial condition that leads to a stable fixed point. This implies that when one selects a period \( p \) orbit their control law will stabilize around one of the possible stable period \( p \) orbits. Which stable periodic orbit emerges is a function of the initial condition.

The update rule Steingrube et al. used for \( \mu^{(p)} \) is a steepest descent update, using the squared \( \ell_2 \)-norm between updates. This is sensible given that the unit matrix is the basis of the linearization. It is important to note that while this is a largely successful strategy, it is by no means guaranteed to work. We next evaluate the performance of their control scheme in simulation.

We begin by observing the dynamics of the system within the chaotic regime (no control). Figure 1 details the dynamics of the 2-dimensional neural control circuit. The top left panels show the output of the sigmoid function, \( x_i(t + 1) \), as a function of \( x_i(t) \), \( i \in 1, 2 \). The top right panel shows the phase plot of \( x_1(t) \) vs. \( x_2(t) \). The bottom, elongated panel plots the time-series of both \( x_1, x_2, \) and \( \mu^{(p)} + 1 \) (to match the range of these variables, \([0,1]\)). The influence of both \( x_1(t) \) and \( x_2(t) \) upon \( x_1(t + 1) \) is apparent from orientation of the sigmoid function in the bottom of the two left panels. The phase portrait clearly illustrates that the system’s dynamics reside within a narrow region of the available phase space. The time-series panel makes clear that in the absence of control, the system exhibits chaotic dynamics.
Figure 1: Chaotic System. top left: $x_i(t+1)$ as a function of $x_i(t)$, $i \in 1, 2$. top right: The phase plot of $x_1(t)$ vs. $x_2(t)$. bottom: The time-series of both $x_1$ (blue), $x_2$ (red), and $\mu^{(p)}+1$ (black). Note translation of $\mu$ to match the range of the variables $[0,1]$

Figure 2 demonstrates the efficiency of this technique by activating the control scheme and setting a desired stable periodic orbit of period 5. The phase portrait is tightly grouped around 5 regions, representing the 5 phases of the periodic orbit, and the time-series panel details the recurring period 5 pattern.
Figure 2: Stabilized period 5 orbit. top left: $x_i(t+1)$ as a function of $x_i(t), i \in 1,2$. top right: The phase plot of $x_1(t)$ vs. $x_2(t)$. bottom: The time-series of both $x_1$ (blue), $x_2$ (red), and $\mu(p) + 1$ (black). Note translation of $\mu$ to match the range of the variables [0,1].

A range of different periods from 2-20 were tested, and the probability of stabilizing a periodic orbit, given the $1/p$ fixed adaptation rate, matched the results of Steingrube et al. 2010. Figure 3 details the stabilization of a period 19 orbit.
Figure 3: Stabilized period 19 orbit. *top left:* $x_i(t + 1)$ as a function of $x_i(t), i \in 1, 2$. *top right:* The phase plot of $x_1(t)$ vs $x_2(t)$. *bottom:* The time-series of both $x_1$ (blue), $x_2$ (red), and $\mu^{(p)} + 1$ (black). Note translation of $\mu$ to match the range of the variables [0,1].

Figure 4 shows the transition from the initially chaotic system, through a period 4 orbit, followed by a stable period 11 orbit. The brief chaotic transient as the system switches from a period 4 stable orbit to a period 11 stable orbit is clear. Note that the control parameter $\mu$ stabilizes at different values for different periodic orbits.
Figure 4: Transition between periodic orbits. top left: $x_i(t + 1)$ as a function of $x_i(t)$, $i \in 1, 2$. top right: The phase plot of $x_1(t)$ vs. $x_2(t)$. bottom: The time-series of both $x_1$ (blue), $x_2$ (red), and $\mu^{(p)} + 1$ (black). Note translation of $\mu$ to match the range of the variables [0,1]

Our simulations validate the efficacy of the control scheme of Steingrube et al. 2010. A few observations are worth noting. First, comparing the simulations and the left panels of Figures 1-4 reveal that the sigmoid functions of $x_i(t + 1)$ stabilized at the position of the uncontrolled system. This was after a transitional period during which they translated along an axis perpendicular to the front of the functions. This was despite the fact that $\mu^{(p)}$ often remained closer to -1 than 0, implying $\Delta_i \to 0$ as $t \to \infty$. Thus, the control asymptotes at the value which stabilizes the system, which in turn maintains stability.

Our analysis, simulations, and observations support Steingrube et al.’s claim that their control scheme enables one to stabilize the periodic orbits of the 2-dimensional neural control circuit over a wide range of periods. This is an impressive feat opening up the possibility that autonomous robots need not require an ever-increasing number of special-
ized circuits for each behavior. They argued that their control scheme is greatly applicable to any chaotic system. They leave out an important caveat, though: dependent upon the system, the number of period $p$ orbits may be large, introducing a stochastic element to the system. Furthermore, they suggested their results might be relevant for understanding how the brain works, which they believe to also be a system capable of producing multiple behavioral outputs without a specific neural module for each. Before we consider the greater implications of this work for robotics in general, let’s dwell upon what Steingrube et al. did with their stable periodic orbits once they have access to them.

**Behavior beyond the lookup table**

There still remains the issue of how to map sensory inputs to the available set of motor outputs. The solution of Steingrube et al. is to build a lookup table from “Environmental stimuli and conditions” to “Behavioral pattern” (Table 1). Thus, while their control scheme allows them to chart trajectories through motor space, the connections with these and sensor space are coded as pointers by the engineer. Why did they choose such a basic solution to this problem? What challenges does the lookup table hide?

First, there is the issue of classifying those sensor inputs that convey concrete information about the environment in which the robot operates. Essentially, their category “Environmental stimuli and conditions” answers the questions, “What is an object?” and “What is out there?” Second, a set of pointers from sensor input to motor output makes the system purely reactive. Consequently, it has no ability to plan or predict. Such a deterministic “if A then B” system is quite limited. While we will not address the complex issue of classifying sensor inputs, our proposal is that probabilistic methods, coupled with a spatial embedding, would engender autonomous robots with the ability to plan and predict.

With the use of probabilistic mappings between sensor inputs and motor outputs, one can relax the overly rigid “if A then B” architecture. The intuition being that the environment is neither deterministic nor chaotic. “The organism commonly faces a world that repeats itself, that is consistent to some degree in obeying laws, that is not wholly chaotic” (W. Ross Ashby 1960 p. 139). Of course, one might argue that probabilistic mappings only create the illusion of flexibility, when in fact the roboticist has only introduced a random element.

It would be foolish to simply plan actions based upon a recursive estimate of the frequencies of occurrence of sensory inputs. For example, knowing that on average 0.00000034% of my sensory inputs are of cats jumping into my lap doesn’t recommend I randomly, but with the same average frequency, wear thick pants to avoid claws. This naive estimate leaves out the notion of context, that is to say, the conditions under which sundry sensory
inputs are more or less likely. We propose that the framework provided by self-localizing
and mapping systems (SLAM) offers a valuable approach for embedding probabilistic
sensorimotor mappings into a spatial context. We will review the mathematics under-
ing SLAM as well as show an example of its use. We then propose how the data structures
involved might be used to store distributions over sensory inputs to aid in motor planning
and prediction.

All SLAM systems operate according to the same basic probabilistic engine, which we
summarize below (taken from Durrant-Whyte and Bailey 2006). At discrete time $k$, we
define the quantities:

- $x_k$: the state vector describing the kinematics of the robot.
- $u_k$: the control vector, applied at time $k-1$ and influencing the state at time $k$.
- $m_i$: a vector describing the $i^{th}$ location of the map, which is assumed to be time invariant.
- $z_{ik}$: an observation of the $i^{th}$ location at time $k$.

With these definitions we can define the following sets:

- $X_{0:k} = \{x_0, x_1, \ldots, x_k\}$: the state history of the robot.
- $U_{0:k} = \{u_0, u_1, \ldots, u_k\}$: the control history.
- $Z_{0:k} = \{z_0, z_1, \ldots, z_k\}$: the observation history.

Now we are ready to define the probabilistic engine for SLAM. At each time $k$, we must
compute the joint distribution,

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0).$$

(11)

This is the joint posterior distribution of the robot’s state and map at time $k$ given its
observation and control histories, as well as its initial state. It is assumed that the robot
has a dynamical model of itself. All SLAM implementations to date assume a known
initial state and observation, allowing for a recursive update from time $k-1$ to time $k$. The
observation model is the probability of making the observation $z_k$ given the current robot
state and map. It captures the ability of the robot to predict its forthcoming sensor inputs as,

$$P(z_k | x_k, m)$$

(12)

and it is generally assumed that the observations are conditionally independent given the
robot state and map.

The motion model describes the state transitions of the robot based upon the dynam-
ical system model it has of itself:

$$P(x_k | x_{k-1}, u_k).$$

(13)
The SLAM update algorithm is then implemented in a two-step process:

**Time-update**

\[
P(x_k, m | Z_{0:k-1}, U_{0:k}, x_0) = \int P(x_k | x_{k-1}, u_k) \times P(x_k, m | Z_{0:k-1}, U_{0:k-1}, x_0) dx_{k-1}
\]

**Measurement Update**

\[
P(x_k, m | Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k | x_k, m) P(x_k, m | Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k | Z_{0:k-1}, U_{0:k})}
\]

This is the most common formulation of the SLAM algorithm, and the one we will focus upon. It assumes a known initial state and support for all the distributions involved. With regards to the map, the robot must *a priori* have some grid, in 2D, or a cubic mesh, in 3D, which it populates with observations.

To validate the performance of the SLAM framework, we recreate the results of Eliazar and Parr 2004. We focused upon their particular SLAM implementation because of the clever, flexible data structures they designed. We used the open source code dp-slam0.1.1 (available at: http://www.cs.duke.edu/~parr/dpslam/). This code came with a log file acquired by a remote control car equipped with a laser range finder. The laser and odometry data were collected as the robot circled the second floor of the computer science department at Duke University. Figure 5 details the complete map created by the DP-SLAM algorithm. The smooth closure of the circular map demonstrates the ability of the algorithm to correct localization errors even for large maps.
Figure 5: Complete map reconstruction via DP-SLAM. This reconstruction was made using the DP-SLAM code of Eliazar and Parr. The map is constructed from odometry and laser range data recorded by a remote control robot circling the interior of the second floor of the computer science department at Duke University.

The rays that project out from the main, circular route of the robot indicates regions of low sample density, leaving behind residuals. Figure 6 shows a detailed local map. The larger map is constructed from a series of these local maps in a hierarchical fashion to further correct for localization errors.
Figure 6: Local map reconstruction via DP-SLAM. This reconstruction was also made using the DP-SLAM code of Eliazar and Parr. This is a one member of a set of local map from which the complete map of the computer science department at Duke University was constructed.
Upon closer consideration of the observation model it becomes clear that there is nothing restricting the observations just to identifying obstructions, as they are generally treated for the purpose of map building. The observation model could be not only a distribution over raw sensor values, but also one over object classes e.g. the probability of seeing a red ball. Likewise, the state $x_k$ would now be updated according to the control $u_k$ associated with the action policy e.g. one of the stable periodic orbits from the lookup table of Steingrube et al. 2010. Hence, the probabilistic mapping between sensor input and motor output becomes indirect, with the direct probabilistic relationship being between predicted sensor inputs and the robot’s location within the map. Furthermore, since a SLAM system has both a model of its own dynamics as well as one of its environment, such a system would be able to not only predict its forthcoming sensor inputs, but also plan trajectories through its environment according to various objectives. This could all be formulated within the principles of optimal control theory (Stengel 1994). The objectives would still have to be supplied by the roboticist, but they could be much more general than those available to a purely reactive machine.

**Discussion**

In this write up, we examined recent ideas in two areas related to autonomous robotics. The first was the use of chaos control to generate multiple, independent motor trajectories using a single neural control circuit. We explored the concepts and mathematics underlying this work in the context of recreating the results of Steingrube et al. 2010. Next we critiqued the restrictive lookup table framework they offered for mapping sensor input to motor output by proposing an expansion of SLAM systems. We examined the probabilistic engine of SLAM and were able to re-construct the map generated by Eliazar and Parr 2004 based upon their software. We argue that taken together, these advances make possible more robust and flexible autonomous robots.

The motor trajectories output by the neural control circuit are intrinsic to the system; the user merely selects the period. This means that any optimization over actions is restricted to the set of periodic orbits the neural control circuit is capable of generating. This is not unlike the situation natural organisms find themselves in: they must base their actions upon the motor outputs available to them. Though, in exchange for this restrictive set of motor outputs, the task of action selection may be executed much more quickly. This is obvious when one compares the situation in which a dog has to choose the speed at which it wants to run versus one in which it must organize its muscles anew into an appropriate gait every time it wishes to run.

Of course our proposal does not go all the way toward the ideal autonomous robot. For one, SLAM systems currently require that the observations at a given map location
are time invariant. This means that dynamic objects, for example roaming predators or prey, can not be accurately represented. Consequently, SLAM systems are currently unable to differentiate sensor inputs induced by self motion from those caused by external forces. In addition, we have assumed away the entire process of filtering and categorizing sensor input. Addressing these functionalities will be left to further work.
Bibliography


MATLAB Code

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Sante Fe 2012 CSSS: John D. Long II and Mikkel Vestergaard%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Project::Topic::Chaotic Control
%%% Example System: from
"Self-organized adaptation of a simple neural circuit enables complex robot behaviour"
%by Silke Steingrube, Marc Timme, Florentin Wörgötter and Poramate Manoonpong

%%% Re-create neural chaos control
clear all
close all
% randomize initial conditions of number generator
rng('shuffle')

%%% Params %%%

%fixed parameters
%bias for neurons 1 and 2 (set according to article)
theta1  = -3.4;
theta2  = 3.8;
%interaction weights (set according to article)
W       = zeros(2,2);
W(1,1)  = -22.0;
W(1,2)  = 5.9;
W(2,1)  = -6.6;
%domain for each variable x
domain  = 0:0.01:1;
Ld      = length(domain);
[d1,d2] = meshgrid(domain,domain); %meshgrid for 3D plot

%%% control inputs for neuron 1 and 2
%period of control
p      = 4;
chaos  = p*50; % length of initial uncontrolled, chaotic activity
%Number of timepoints to run
Nt     = p*300;
%adaptation rate (heuristic set in article)
lambda  = 1/p;
%control weight
mu     = 0;
MU     = zeros(Nt+1,1);
%switch for period change
dp     = 0;
%initial control values
c1     = 0;
c2     = 0;

%%%%%%%%%%%%%%%%
%%% Functions %%%

%sigmoid functions for neurons 1 and 2
x1_t1 = @(x1t0) 1./(1+exp(-x1t0));
x2_t1 = @(x2t0) 1./(1+exp(-x2t0));
%% Initial conditions

\[ temp1 = \text{rand}; \]
\[ temp2 = \text{rand}; \]
\[ x1_{t0} = x1_{t1}(theta1 + W(1,:)*[temp1;temp2] + c1); \]
\[ x2_{t0} = x2_{t1}(theta2 + W(2,:)*[temp1;temp2] + c2); \]

% Initialize output vectors starting at time zero
\[ X1 = \text{zeros}(Nt+1,1); \]
\[ X1(1) = x1_{t0}; \]
\[ X2 = \text{zeros}(Nt+1,1); \]
\[ X2(1) = x2_{t0}; \]

%% figure handles

% Figure layout
\[ \text{scrnz} = \text{get}(0, 'screensize'); \]
\[ \text{figure('Position', scrnz, 'color', 'w')}; \]

% X1 surface
\[ \text{ax1} = \text{subplot(3,3,4)}; \]
\[ \text{title('bfX1_{(t+1)}'), fontsize, ll, 'fontname', 'arial')}; \]
\[ \text{xlabel('X1_{(t)}'), fontsize, ll, 'fontname', 'arial')}; \]
\[ \text{yspace([0 1 0 1])}, \]
\[ \text{axis square}, \]
\[ \text{view([0 90])}, \]
\[ \text{grid off, hold on}, \]
\[ \text{colorbar}; \]

% X2 surface
\[ \text{ax2} = \text{subplot(3,3,1)}; \]
\[ \text{title('bfX2_{(t+1)}'), fontsize, ll, 'fontname', 'arial')}; \]
\[ \text{xlabel('X1_{(t)}'), fontsize, ll, 'fontname', 'arial')}; \]
\[ \text{yspace([0 1 0 1])}, \]
\[ \text{axis square}, \]
\[ \text{view([0 90])}, \]
\[ \text{grid off, hold on}, \]
\[ \text{colorbar}; \]

% X1 vs X2 phase portrait
\[ \text{ax3} = \text{subplot(3,3,[2 3 5 6])}; \]
\[ \text{title('bfX1 vs X2: phase plot')}; \]
\[ \{\text{sprintf('Target Period: %d',p)}\}], \]
\[ \text{fontsize, ll, 'fontname', 'arial'}]; \]
\[ \text{xlabel('X1', 'fontsize', ll, 'fontname', 'arial')}; \]
\[ \text{ylabel('X2', 'Rotation', 0, 'fontsize', ll, 'fontname', 'arial')}; \]
\[ \text{axis([0 1 0 1])}, \]
\[ \text{axis square}, \]
\[ \text{hold on}; \]
\[ \text{colorbar}; \]

% X1 and X2 timeseries
\[ \text{ax4} = \text{subplot(3,3,[7 8 9])}; \]
\[ \text{title('bfX1_{(blue)}, X2_{(red)}, and \mu^{(p)}_{black}')}, \]
\[ \{\text{'time-series'}\}], \]
\[ \text{'fontsize', ll, 'fontname', 'arial'}]; \]
\[ \text{ylabel('Network Output', 'fontsize', ll, 'fontname', 'arial')}; \]
\[ \text{xlabel('Time (steps)', 'fontsize', ll, 'fontname', 'arial')}; \]
\[ \text{axis([0 Nt 0 1])}, \]
\[ \text{hold on}; \]
\[ \text{h = zoom}; \]
set(h,'Motion','horizontal','Enable','on');

% Overlay mu axis
ax5 = axes('Position',get(ax4,'Position'),...
        'YAxisLocation','right',...
        'Color','none',...
        'YColor','k');
ylabel('
\mu + 1', 'Rotation', 270, 'fontsize', 11, 'fontname', 'arial'),...
axis([0 Nt 0 1]), hold on
linkaxes([ax4 ax5], 'xy')

% Plot handles for marginal functions and fixed point line
% x1
X1_t1 = reshape(x1_t1(theta1 + W(1,:)*[d1(:) d2(:)]' + c1),Ld,Ld);
x1_fn = surf(d1,d2,X1_t1,...
    'facecolor','interp',...
    'linestyle','none',...
    'parent',ax1);

% plot initial point on 3D surface
x1_pt = plot3(x1_t0,x2_t0,x1_t0,...
    'parent',ax1,...
    'marker','o',...
    'color','k',...
    'markerfacecolor','k',...
    'markersize',5);

% x2
X2_t1 = reshape(x2_t1(theta2 + W(2,:)*[d1(:) d2(:)]' + c2),Ld,Ld);
x2_fn = surf(d1,d2,X2_t1,...
    'facecolor','interp',...
    'linestyle','none',...
    'parent',ax2);

% plot initial point on 3D surface
x2_pt = plot3(x1_t0,x2_t0,x2_t0,...
    'parent',ax2,...
    'marker','o',...
    'color','k',...
    'markerfacecolor','k',...
    'markersize',5);

% Phase portrait of X1 against X2
x1x2 = plot(X1(1),X2(1),...
    'parent',ax3,...
    'linestyle','none',...
    'marker','o',...
    'markersize',3,...
    'markeredgecolor','k',...
    'markerfacecolor','k');

% X1 and X2 time-series
x1ts = plot(0,X1(1),...
    'parent',ax4,...
    'marker','o',...
    'color','b',...
    'markerfacecolor','b',...
    'markersize',3);

x2ts = plot(0,X2(1),...
    'parent',ax4,...
    'marker','o',...
    'color','r',...
    'markerfacecolor','r',...
% index ii is for t->t+1
for ii = 1:Nt
    % period switching condition
    if ii == 600
        p = 11;
        lambda = 1/p;
        dp = 0;
    end

    %Do we have a period p sample?
    if p > 0 && ii > p+chaos-1 && ((mod(ii,p+1)) == 0)
        %Did the period just switch?
        if dp < 1
            %flip switch to 1
            dp = 1;
            %set mu to initial value of -1
            mu = -1;
            %Calculate difference between current and previous period value
            delta1 = X1(ii) - X1(ii-p);
            delta2 = X2(ii) - X2(ii-p);
            %update control values
            c1 = mu*W(1,:)*[delta1;delta2];
            c2 = mu*W(2,:)*[delta1;delta2];
            %update mu according to adaptation
            mu = mu + lambda*((delta1^2+delta2^2)/p);
        end
    else
        %Calculate difference between current and previous period value
        delta1 = X1(ii) - X1(ii-p);
        delta2 = X2(ii) - X2(ii-p);
        %update control values
        c1 = mu*W(1,:)*[delta1;delta2];
        c2 = mu*W(2,:)*[delta1;delta2];
        %update mu according to adaptation
        mu = mu + lambda*((delta1^2+delta2^2)/p);
    end

    % update map functions with control %
    ud_x1 = x1_t1(theta1 + W(1,:)*[x1_t0;x2_t0] + c1);
    ud_x2 = x2_t1(theta2 + W(2,:)*[x1_t0;x2_t0] + c2);

    % update variables
    X1(ii+1) = ud_x1;
    X2(ii+1) = ud_x2;
    MU(ii+1) = mu+1;
else
%%% update map functions without control %%%
ud_x1 = x1_t1(theta1 + W(1,:)*[x1_t0;x2_t0]);
ud_x2 = x2_t1(theta2 + W(2,:)*[x1_t0;x2_t0]);

%update variables
X1(ii+1) = ud_x1;
X2(ii+1) = ud_x2;
MU(ii+1) = mu+1;
end

%update plots
set(x1_pt,'xdata',x1_t0,'ydata',x2_t0,'zdata',ud_x1);
set(x2_pt,'xdata',x1_t0,'ydata',x2_t0,'zdata',ud_x2);
set(x1x2,'xdata',X1(1:ii+1),'ydata',X2(1:ii+1));
set(x1ts,'xdata',0:ii,'ydata',X1(1:ii+1));
set(x2ts,'xdata',0:ii,'ydata',X2(1:ii+1));
set(muts,'xdata',0:ii,'ydata',MU(1:ii+1));
X1_t1 = reshape(x1_t1(theta1 + W(1,:)*[d1(:) d2(:)]' + c1),Ld,Ld);
set(x1_fn,'zdata',X1_t1)
X2_t1 = reshape(x2_t1(theta2 + W(2,:)*[d1(:) d2(:)]' + c2),Ld,Ld);
set(x2_fn,'zdata',X2_t1)

%update time step
x1_t0 = ud_x1;
x2_t0 = ud_x2;
% pause(.1)
end

set(gcf,'color','white','PaperPositionMode','auto')