B. TECH. PROJECT REPORT

On

Development of Augmented environment through sensor data using Machine learning Models

BY

Himanshu Meena

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

DEC 2017

Development of Augmented environment through sensor data using Machine learning Models

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in

ELECTRICAL ENGINEERING

Submitted by:
Himanshu Meena

Guided by:
Dr.Abhishek Srivastava
(Assistant Professor)

INDIAN INSTITUTE OF TECHNOLOGY INDORE
5th December, 2017

CANDIDATE'S DECLARATION

We hereby declare that the project entitled "**Development of Augmented environment through sensor data using Machine learning Models"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in Electrical Engineering completed under the supervision of Dr. Abhishek Srivastava Assistant Professor Computer Science and Engineering**,** IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student with date

CERTIFICATE by BTP Guide

It is certified that the above statement made by the students is correct to the best of my knowledge.

Signature of BTP Guide with dates and their designation

Preface

This report on "**Development of Augmented environment through sensor data using Machine learning Models"** is prepared under the guidance of **Dr. Abhishek Srivastava**.

Through this report we have tried to give a detailed design of the model to record, analyse and implement data from various environment sensors and create a virtual environment inside Second Life. The best fit model is from implementing algorithm on data from various sources and most fit model is used to predict the future values of different environment variables that in turn are configured to be used to form virtual environment inside Second Life World.

We have tried to the best of our abilities and knowledge to explain the content in a lucid manner. We have also added interactive models and figures to make it more illustrative.

Himanshu Meena
B.Tech. IV Year
Discipline of Electrical
IIT Indore

Acknowledgements

I would like to thank my B.Tech Project supervisor Dr. Abhishek Srivastava for his constant support in structuring the project and his valuable feedback throughout the course of this project. He gave me an opportunity to discover and work in such an interesting domain. His guidance proved really valuable in all the difficulties I faced in the course of this project.

I am really thankful to my BTP partner Abishek Kumar for his contribution and support throughout the project. During this journey we faced a lot of problems but since I had such an amazing partner those high hurdles didn't seem that high and with mutual help we completed this project together with flying colors.

I am also thankful to my family members,friends and colleagues who were a constant source of motivation. I am really grateful to Dept. of Computer Science & Engineering & Dept. of Electrical Engineering, IIT Indore for providing with the necessary hardware utilities to complete the project. I offer sincere thanks to everyone who else who knowingly or unknowingly helped me complete this project.

Himanshu Meena
140002013
Discipline of Electrical Engineering
Indian Institute of Technology Indore

Abstract

Second Life is an online virtual world,developed and owned by the San Francisco-based firm Linden Lab. The platform principally features 3D-based user-generated content. The secondlife environment is scripted in Linden Scripting Language (LSL). The climatic parameters received as a response JSON is manipulated and optimized in this end. The script corresponding to any object in secondlife is constrained to it in a limited radius. Thus a deep study in variation of factors were made and scripted accordingly to get a more close weather conditions as that of in the real world. We are building a virtual environment that could mimic real characteristics and implement machine learning model for futuristic parameters.

Table of Contents

Chapter 1

Introduction

This chapter highlights the background and motivation for the project. The problem statement of the project has been described and the importance of the results is also clearly portrayed. Towards the end of the chapter the objectives and expectation from this project are also outlined.

1.1 Background

Second Life is an online virtual world, developed and owned by the San Francisco-based firm Linden Lab . The platform principally features 3D-based user-generated content. Second Life also has its own virtual currency, the Linden Dollar, which is exchangeable with real world currency. Users can explore the world (known as the grid), meet other residents, socialize, participate in individual and group activities, build, create, shop, and trade virtual property and services with one another. Built into the software is a 3D modeling tool based on simple geometric shapes, that allows residents to build virtual objects. There is also a procedural scripting language, Linden Scripting Language, which can be used to add interactivity to objects. Sculpted prims(sculpties), mesh, textures for clothing or other objects, animations, and gestures can be created using external software and imported. The Second Life terms of service provide that users retain copyright for any content they create, and the server and client provide simple digital rights management (DRM) functions. With 900,000 active users a month, who get payouts of $60 million in real-world money every year.

If we see the rate at which humans are technologically advancing, the time when humans won't even have to move is not far away and I think virtual world if the best way to realize that dream.Second life is, as its name suggest, a second virtual world with infinite possibilities. Second Life is used as a platform for education by many institutions, such as colleges, universities, libraries and government entities.The University of San Martin de Porres of Peru has been developing Second Life prototypes of Peruvian archeological buildings, and training teachers for this new paradigm of education.Second Life is used for scientific research, collaboration, and data visualization.It also gives companies the option to create virtual workplaces to

allow employees to virtually meet, hold events, practice any kind of corporate communications, conduct training sessions in 3D immersive virtual learning environment, simulate business processes, and prototype new products.

In our project we are contributing in creating a virtual environment based on real time environment variable and stepping another step towards making second life as real and as close to real world as possible.

1.2 Objective

As per the above discussion, our main objective is to the Development of Augmented environment through sensor data using Machine learning Model for implementing the following in secondlife . The project has been divided into different modules:

- To extract weather data parameter values from physical sensors through arduino nodes
- To design a machine learning model to make it handle and process the sensor data
- To enhance the model and implement an algorithm to make it sustain on its own
- To test and compare the predictions from the model with the real values
- To develop a virtual environment in Second life from the sensor data acquired

Chapter 2

Model Overview

This chapter discusses on various concept used n this project . The chapter starts with the discussion on ARIMA model followed by explanation of how environment parameters are handled in secondlife. the end of the chapter comprises of discussion on flask, its features and advantages.

2.1 ARIMA

ARIMA models provide another approach to time series forecasting. Exponential smoothing and ARIMA models are the two most widely-used approaches to time series forecasting, and provide complementary approaches to the problem. While exponential smoothing models were based on a description of trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data.

Before we introduce ARIMA models, we need to first discuss the concept of stationarity and the technique of differencing time series. Then its composite parts AR(AutoRegressive) and MA(Moving Average) models are discussed in next subsection and then ARIMA is discussed.

2.1.1 Stationarity and differencing

Stationarity

A stationary time series is one whose properties do not depend on the time at which the series is observed. So time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any period of time.

Some cases can be confusing — a time series with cyclic behaviour (but not trend or seasonality) is stationary. That is because the cycles are not of fixed length, so before we observe the series we cannot be sure where the peaks and troughs of the cycles will be.

In general, a stationary time series will have no predictable patterns in the long-term. Time plots will show the series to be roughly horizontal (although some cyclic behaviour is possible) with constant variance.

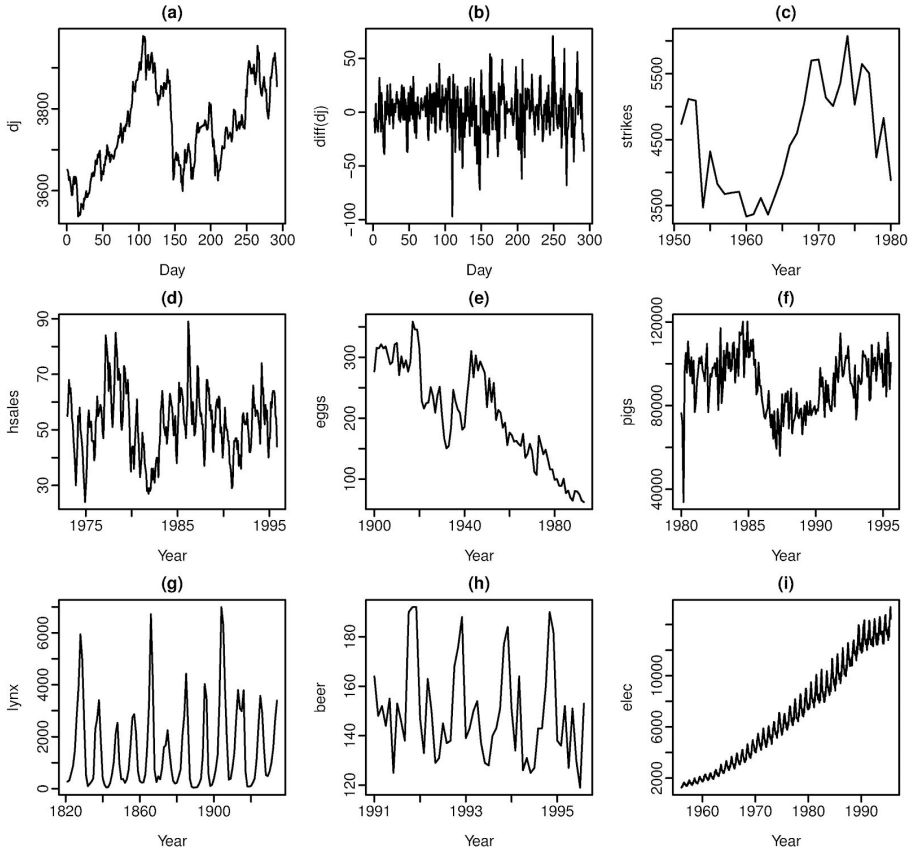


fig: Which of these series are stationary? (a) Dow Jones index on 292 consecutive days; (b) Daily change in Dow Jones index on 292 consecutive days; (c) Annual number of strikes in the US; (d) Monthly sales of new one-family houses sold in the US; (e) Price of a dozen eggs in the US (constant dollars); (f) Monthly total of pigs slaughtered in Victoria, Australia; (g) Annual total of lynx trapped in the McKenzie River district of north-west Canada; (h) Monthly Australian beer production; (i) Monthly Australian electricity production.(Trend rules out series (a), (c), (e), (f) and (i). Increasing variance also rules out (i). That leaves only (b) and (g) as stationary series.)

Differencing

In Figure 2.1, notice how the Dow Jones index data was non-stationarity in panel (a), but the daily changes were stationary in panel (b). This shows one way to make a time series stationary — compute the differences between consecutive observations. This is known as differencing.

Transformations such as logarithms can help to stabilize the variance of a time series. Differencing can help stabilize the mean of a time series by removing changes in the level of a time series, and so eliminating trend and seasonality.

As well as looking at the time plot of the data, the ACF plot is also useful for identifying non-stationary time series. For a stationary time series, the ACF will drop to zero relatively quickly, while the ACF of non-stationary data decreases slowly. Also, for non-stationary data, the value of r1 is often large and positive.

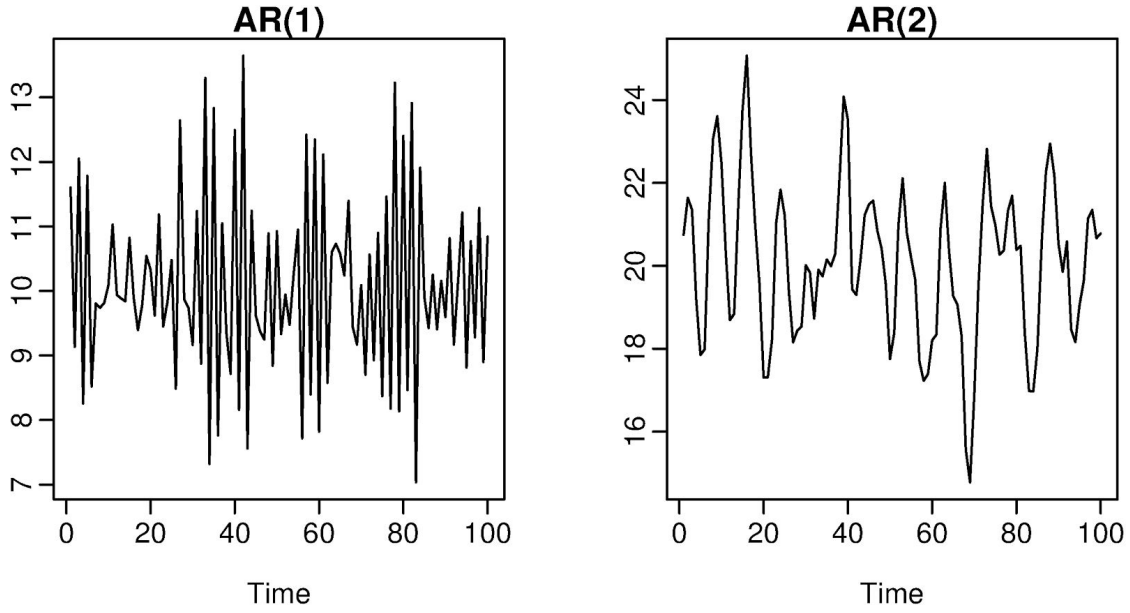
2.1.2 Autoregressive models

In a multiple regression model, we forecast the variable of interest using a linear combination of predictors. In an autoregression model, we forecast the variable of interest using a linear combination of *past values of the variable*. The term *auto*regression indicates that it is a regression of the variable against itself.

Thus an autoregressive model of order **p** can be written as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

where c is a constant and ε_t is white noise. This is like a multiple regression but with *lagged values* of y_t as predictors. We refer to this as an AR(p) model. Autoregressive models are remarkably flexible at handling a wide range of different time series patterns.



$y_t = 18 - 0.8 y_{t-1} + \varepsilon_t$ $y_t = 8 - 1.3 y_{t-1} - 0.7 y_{t-2} + \varepsilon_t$

Fig: Two examples of data from autoregressive models with different parameters. Left: AR(1) . Right: AR(2) with In both cases, ε_t *is normally distributed white noise with mean zero and variance on*

2.1.2 Moving average models

Rather than use past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model.

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q}$$

where ε_t is white noise. We refer to this as an MA(q) model. Of course, we do not *observe* the values of ε_t , so it is not really regression in the usual sense. Notice that each value of y_t can be thought of as a weighted moving average of the past few forecast errors.

Relationship between AR and MA models

It is possible to write any stationary AR(p) model as an MA(∞) model. For example, using repeated substitution, we can demonstrate this for an AR(1) model :

$$y_t = \phi_1 y_{t-1} + \varepsilon_t$$

$$= \phi_1(\phi_1 y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t$$

$$= \phi_1^2 y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t$$

$$= \phi_1^3 y_{t-3} + \phi_1^2 \varepsilon_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t$$

etc.

So eventually we obtain:

$$y_t = \varepsilon_t + \phi_1 \varepsilon_{t-1} + \phi_1^2 \varepsilon_{t-2} + \phi_1^3 y_{t-3} + \ldots,$$

an MA(∞) process.

The reverse result holds if we impose some constraints on the MA parameters. Then the MA model is called "invertible". That is, that we can write any invertible MA(q) process as an AR(∞) process.

2.1.3 ARIMA models

If we combine differencing with autoregression and a moving average model, we obtain a non-seasonal ARIMA model. ARIMA is an acronym for AutoRegressive Integrated Moving Average model ("integration" in this context is the reverse of differencing). The full model can be written as

$$y'_t = c + \phi_1 y'_{t-1} + \ldots + \phi_p y'_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q}$$

where y'_t is the differenced series (it may have been differenced more than once). The "predictors" on the right hand side include both lagged values of y_t and lagged errors. We call this an ARIMA(p,d,q) model, where

- p = order of the autoregressive part,
- d = degree of first differencing involved,
- q = order of the moving average part.

The same stationarity and invertibility conditions that are used for autoregressive and moving average models apply to this ARIMA model.

We start combining components in this way to form more complicated models. Selecting appropriate values for p, d and q can be difficult. To select appropriate values for p, d, q what we do actually is finding best with the help of certain algorithms. In this project what we used is Square Mean Error and Maximum

Likelihood Estimation along with Kalman filter Algorithm. In the next section they explained in detail.

2.1.4 Maximum Likelihood Estimation

ARIMA first make data stationary by differencing or using functions such as logarithmic, square root, cube root etc. By doing this we have successfully found value of d but for p and q you need to find AutoCorrelation Function(ACF) and Partial AutoCorrelation Function(PACF) of the dataset with itself. Even by doing all this you arrive at a range of values from which you need to find the best fit for the model that can be done by various methods and one of them is differentiating and evaluating them on the basis of mean square error. But to find this error you need to first find the coefficients of AR and MA models. You can achieve it by using Kalman filter algorithm on maximum likelihood estimation.

In this section, I describe the algorithm used to compute the maximum likelihood estimates of the ARMA(p,q) process. Suppose that we want to fit the (mean zero) time series to the the following ARMA(p,q) model

$$y_t = \phi_1 y_{t-1} + ... + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + ... + \theta_q \varepsilon_{t-q}$$

where ε_t is an i.i.d. shock normally distributed with mean zero and variance σ^2. Let r be max(p, q + 1), and rewrite the model as

$$y_t = \phi_1 y_{t-1} + ... + \phi_r y_{t-r} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + ... + \theta_{r-1} \varepsilon_{t-r+1}$$

We interpret $\phi_j = 0$ for $j > p$ and $\theta_j = 0$ for $j > q$.

The estimation procedure is based on the Kalman Filter. To use the Kalman Filter we need to write the model in the following (state-space) form

$$x_{t+1} = Ax_t + R\varepsilon_{t+1}$$

$$y_t = Z' x_t$$

where x_t is an r x 1 state vector, **A** is an *r* x *r* matrix, and **R** and **Z** are *r x 1* vectors. These matrices and vectors are defined as follows

$$\mathbf{A} = \begin{bmatrix} \phi_1 & 1 & 0 & 0 & \cdots & 0 \\ \phi_2 & 0 & 1 & 0 & \cdots & 0 \\ \phi_3 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \ddots & \\ \phi_{r-1} & 0 & 0 & 0 & \cdots & 1 \\ \phi_r & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}; \quad \mathbf{R} = \begin{bmatrix} 1 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \\ \theta_{r-1} \end{bmatrix}; \quad \mathbf{Z} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ \\ 0 \end{bmatrix}$$

To see that the system (3) and (4) is equivalent to (2), write the last row of (3) as

$$x_{r,t-1} = \phi_1 x_{1,t} + \theta_{r-1}\varepsilon_{t+1}$$

Lagging this equation r - 1 periods we find

$$x_{r,t-r+2} = \phi_1 L^{r-1} x_{1,t} + \theta_{r-1} L^{r-1}\varepsilon_{t+1}$$

where we define $L^r x_t = x_{t-r}$ as the r lag operator for any integer $r \geq 0$. The second to last row implies

$$x_{r-1,t+1} = \phi_{r-1} x_{1,t} + x_{r,t} + \theta_{r-2}\varepsilon_{t+1}$$

Lagging *r* x *2* periods we obtain

$$x_{r-1,t-r+3} = \phi_{r-1} L^{r-2} x_{1,t} + x_{r,t-r+2} + \theta_{r-2} L^{r-2}\varepsilon_{t+1}$$

Introducing (5) into the previous equation we find

$$x_{r-1,t-r+3} = \phi_{r-1} L^{r-2} x_{1,t} + \phi_r L^{r-1} x_{1,t} + \theta_{r-1} L^{r-1}\varepsilon_{t+1} + \theta_{r-2} L^{r-2}\varepsilon_{t+1}$$

Take now row *r - 2*,

$$x_{r-2,t+1} = \phi_{r-2} x_{1,t} + x_{r-1,t} + \theta_{r-3}\varepsilon_{t+1}$$

Lagging *r - 3* periods we find

$$x_{r-2,t-r+4} = \phi_{r-2} L^{r-3} x_{1,t} + x_{r-1,t-r+3} + \theta_{r-3} L^{r-3}\varepsilon_{t+1}$$

Plugging (6) into the previous equation we obtain

$$x_{r-2,t-r+4} = [\phi_{r-2} L^{r-3} + \phi_{r-1} L^{r-2} + \phi_r L^{r-1}] x_{1,t} + [\theta_{r-1} L^{r-1} + \theta_{r-2} L^{r-2} + \theta_{r-3} L^{r-3}]\varepsilon_{t+1}$$

Following this iterative procedure until row *r - 1* we find

$$x_{1,t+1} = [\phi_1 + ... + \phi_{r-2} L^{r-3} + \phi_{r-1} L^{r-2} + \phi_r L^{r-1}] x_{1,t} + [\theta_{r-1} L^{r-1} + \theta_{r-2} L^{r-2} + \theta_{r-3} L^{r-3} + ... + 1]\varepsilon_{t+1}$$

or

$$(1 - \phi_1 L^1 - \phi_2 L^2 - ... - \phi_r L^r) x_{1,t+1} = [\theta_{r-1} L^{r-1} + \theta_{r-2} L^{r-2} + \theta_{r-3} L^{r-3} + ... + 1]\varepsilon_{t+1}$$

Now, the observation equation (4) and the definition of **Z** imply

$$y_t = x_{1,t}$$

Using (7) evaluated at t we arrive at the ARMA representation (2),

$$(1 - \phi_1 L^1 - \phi_2 L^2 - ... - \phi_r L^r) y_{1,t} = [\theta_{r-1} L^{r-1} + \theta_{r-2} L^{r-2} + \theta_{r-3} L^{r-3} + ... + 1]\varepsilon_t$$

which proves that the system (3), (4) is equivalent to (2).

Denote by $\hat{x}_{t+1|t} = E_t[x_{t+1}|y_0, ..., y_t; x_0]$ the expected value of x_{t+1} conditional on the history of observations $(y_0, ..., y_t)$. The Kalman Ölter provides an algorithm for computing recursively $\hat{x}_{t+1|t}$ given an initial value $\hat{x}_{1|0} = \bar{0}$. (Note that $\bar{0}$ is the unconditional mean of x_t). Associated with each of these forecasts is a mean squared error matrix, defined as

$$P_{t+1|t} = E[(x_{t+1} - \hat{x}_{t+1|t})(x_{t+1} - \hat{x}_{t+1|t})']$$

Given the estimate $\hat{x}_{t+1|t}$, we use (4) to compute the innovations

$$\begin{aligned} a_t &= y_t - E_t[x_{t+1}|y_0, ..., y_t; x_0] \\ &= y_t - Z'\hat{x}_{t|t-1} \end{aligned}$$

The innovation variance, denoted by w_t, satisfies

$$\begin{aligned} w_t &= E[(y_t - Z'\hat{x}_{t|t-1})(y_t - Z'\hat{x}_{t|t-1})'] \\ &= E[(Z'x_t - Z'\hat{x}_{t|t-1})(Z'x_t - Z'\hat{x}_{t|t-1})'] \\ &= Z'P_{t|t-1}Z \end{aligned}$$

In addition to the estimates $\hat{x}_{t|t-1}$, the Kalman Ölter equations imply the following evolution of the matrices $P_{t+1|t}$

$$P_{t+1|t} = A[P_{t|t-1} - P_{t|t-1}ZZ'P_{t|t-1}/w_t]A' + RR'\sigma^2$$

Given an initial matrix $P_{1|0} = E(x_t x_t')$ and the initial value $\hat{x}_{1|0} = \bar{0}$, the likelihood function of the observation vector $\{y_0, ..., y_t\}$ is given by

$$L = \prod_{t=1}^{T} (2\pi w_t)^{-1/2} exp(-\frac{a_t^2}{2w_t})$$

Taking logarithms, dropping the constant 2, and multiplying by 2 we obtain

$$l = -\sum_{t=1}^{T}[ln(w_t) + a_t^2/w_t]$$

In principle, to find the MLE estimates we maximize (10) with respect to the parameters ϕ_j, θ_j *and* σ^2. However, the following trick allows us to 'concentrate-out' the term σ^2, and maximize only with respect to the parameters ϕ_j *and* θ_j. Suppose we initialize the filter with the matrix $\overline{P_{1|0}} = \sigma^2 P_{1|0}$. Then, from (9) it follows that each $P_{t+1|t}$ is proportional to σ^2, and from (8) it follows that the innovation variance is also proportional to σ^2. This implies that we can optimize first with respect to σ^2 'by hand', replace the result into the objective function, and then optimize the resulting expression (called the 'concentrated log-likelihood') with respect to the parameters ϕ_j, θ_j: To see this, note that (10) becomes

$$l = -\sum_{t=1}^{T}[ln(\sigma^2 w_t) + a_t^2/w_t\sigma^2]$$

and σ^2 is cancelled out in the evolution equations of $P_{t+1|t}$ and in the projections $\hat{x}_{t+1|t}$. So we can directly optimize (11) with respect to σ^2 to obtain

$$\sigma^2 = \frac{1}{T}\sum_{t=1}^{T} a_t^2/w_t$$

Replacing this result into (11) we obtain the concentrated log-likelihood function

$$l = -\sum_{t=1}^{T}[ln(\sigma^2) + ln(w_t) + a_t^2/w_t\sigma^2]$$

$$l = -[\sum_{t=1}^{T} ln(\frac{1}{T}\sum_{t=1}^{T} a_t^2/w_t) + \sum_{t=1}^{T} ln(w_t) + \frac{\sum_{t=1}^{T} a_t^2/w_t\sigma^2}{\frac{1}{T}\sum_{t=1}^{T} a_t^2/w_t\sigma^2}]$$

$$l = -[T ln(1/T) + T + T ln \sum_{t=1}^{T} a_t^2/w_t + \sum_{t=1}^{T} ln(w_t)]$$

or, dropping irrelevant constants,

$$l = -[T + T ln \sum_{t=1}^{T} a_t^2/w_t + \sum_{t=1}^{T} ln(w_t)]$$

Because the innovations a_t and the variances w_t are nonlinear functions of the parameters [ϕ , θ], we use numerical methods to maximize (12).

2.2 SecondLife Environment

In Secondlife users create everything by themselves from a simple pole flag to aeroplane, from their clothes to their houses, from their hairs to their vehicles. So to do that they need some built-in function from Secondlife creator company Linden labs so that they can make changes to their environment. In our project, we have mainly used llhttprequest for taking data from ngrok, llparticlesystem to create particle shower and transformed them to snow or rain and RestrainedloveAPI to change environment with the help of windlight settings.

2.2.1 llHttpRequest

Function: key llHTTPRequest(string url, list parameters, string body **);**

Sends an HTTP request to the specified URL with the body of the request and parameters.

Returns a handle (a key) identifying the HTTP request made.

- string url - a valid HTTP/HTTPS URL
- list parameters - configuration parameters, specified as HTTP_* flag-value pairs[parameter1, value1, parameter2, value2, ..., parameterN, valueN]
- string body - Contents of the request

2.2.1 llParticleSystem

Function: llParticleSystem(list rules **);**

Defines a particle system for the containing prim based on a list of rules.

- list rules - Particle system rules list in the format [rule1, data1, rule2, data2 . . . rule**n**, data**n**]

Defines a particle system that sets the state of the particle emitter within the prim that contains the script. Any other scripts, in the same prim, that call this function will modify the state of the same particle emitter. As such, the particle system defined by this function is a prim property, just like its size, shape, color, etc.

Each prim has only one (1) particle emitter, located at its geometric center, and aligned along the prim's local Z-axis, pointing in the positive Z direction.

This is the one of the only functions which alters the state of the prim's particle emitter; thus, if you wish to change the emitter to a different state (i.e., emitting a different particle system entirely, or shut off the emitter completely), just call the function with the parameters of the new particle system you wish to render instead. Specifying an empty list (i.e., llParticleSystem([]);) turns the emitter off.

Particles are essentially 2D sprites and are always rendered facing the viewer's camera (except when PSYS_PART_RIBBON_MASK is enabled).

The rule / data values are defined can accessed from Appendix A.

2.2.2 RestrainedLoveAPI

The RestrainedLove viewer executes certain behaviours when receiving special messages from scripts in-world. These messages are mostly calls to the llOwnerSay() LSL function.

The RestrainedLove viewer intercepts every llOwnerSay message sent to the viewer. Lines that begin with an at-sign ('@') are parsed as RLV commands. Other lines are forwarded to the user in the Local Chat window, as usual. For instance, a call to llOwnerSay ("@detach=n") sends the detach command with parameter n to the viewer on behalf of the object running the script.

For list of commands please go to Appendix B.

2.3 Server-Side setup

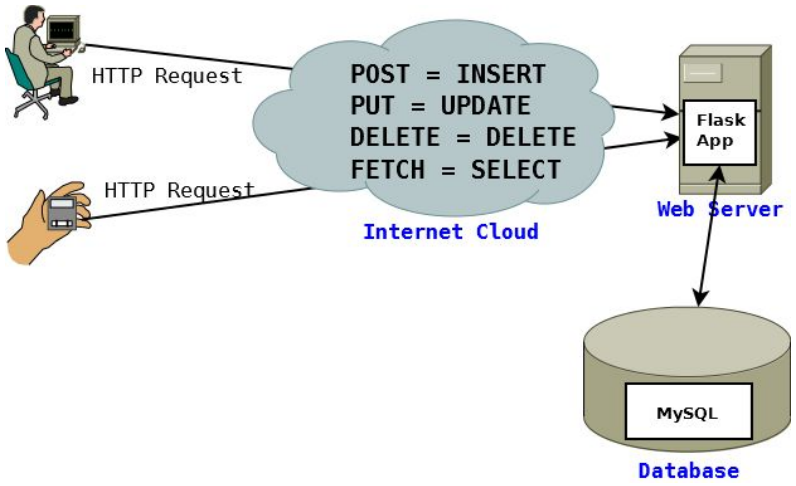
For the purpose of building a real-time streaming environment and database in a fixed and secure environment, we implement a server system.

2.3.1 Flask

Flask is a Python web framework built with a small core and easy-to-extend philosophy.Flask is considered more Pythonic than Django because Flask web application code is in most cases more explicit. Flask is easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running.

Key Features of Flask:

- Contains development server and debugger
- Integrated support for unit testing
- RESTful request dispatching
- Uses Jinja2 templating
- Support for secure cookies (client side sessions)
- 100% WSGI 1.0 compliant
- Unicode-based
- Extensive documentation
- Google App Engine compatibility
- Extensions available to enhance features desired

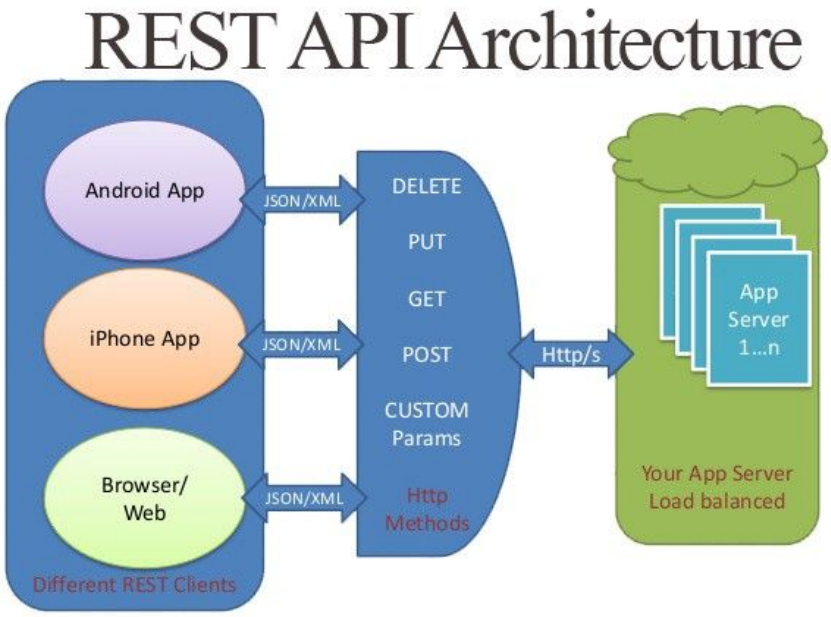


In this project, we have used flask to fire up a server that could stream the Real-time sensor data, as well as the processed data that has been updated by the ARIMA model database.It also supports different formats of data streaming (JSON in this case) and is being running in the server's side.

2.3.2. RESTful APIs

Representational state transfer (REST) or RESTful web services are a way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.By using a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running.Web service APIs that adhere to the REST architectural constraints are called RESTful APIs.[13] HTTP-based RESTful APIs are defined with the following aspects:

- base URL, such as http://api.example.com/resources
- an internet media type that defines state transition data elements (e.g., Atom, microformats, application/vnd.collection+json:91–99 etc.) The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URL or as complex as a Java applet.
- standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE)



For the purpose of this project, we choose REST APIs to maintain consistency and reliability in the data. We choose JSON (Javascript Object notation) format for the purpose of streaming hourly data to Secondlife. In the case of database, CSV formats are being followed (Comma separated values).

2.3.3 NGROK

Ngrok is a handy tool and service that allows you tunnel requests from the wide open Internet to your local machine when it's behind a NAT or firewall. This is useful in a number of cases, such as when you want to test out an add-on you've been writing for HipChat or a custom webhook endpoint for Bitbucket, but you haven't yet deployed your code to an Internet accessible host or PaaS. The most common usage of ngrok sets up a tunnel to localhost using the random hostname ngrok provides by default, e.g., 5a3e3614.ngrok.com.

In this project, the traffic hosted from the server's end using FLASK is being directed and hosted globally in the internet using this program. As FLASK is hosted behind a NAT and the secondlife environment does not lie in the same domain, this has been implemented as a key feature.

Chapter 3

Setup & Implementation

This chapter discusses on the step-by-step setup and implementation of the above explained model in depth on different levels.

3.1 Sensor Network

For the purpose of this project, different sensor datas from several arduino nodes are extracted and streamed into a database as discussed in chapter 2. The format of the database was maintained to be CSV carrying hourly data of a particular day.

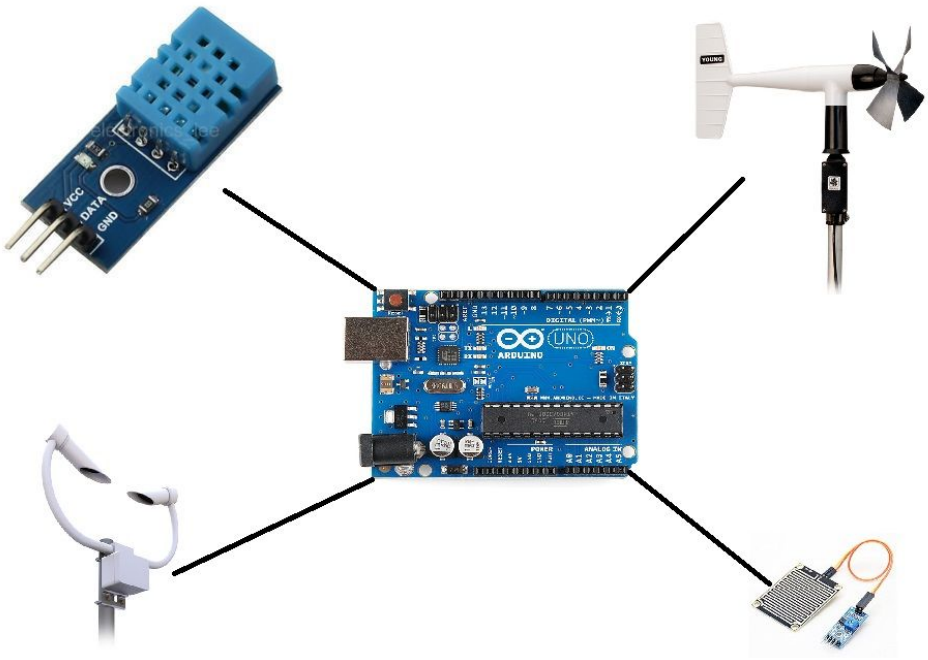
From several different environment variables we have chosen five major environmental factors that can be used to effectively mimic the real world into the Second life environment. These five environment variables are temperature, precipitation, wind speed, wind direction and visibility. They are acquired with the help of different sensors which in themselves different units and are connected through the arduino nodes. Sensors used for each variables are:

- Temperature - DHT11 sensor chips
- Wind Speed - Anemometer
- Wind Direction - Anemometer
- Precipitation - WS100 sensor module
- Visibility - VS2K sensor module

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the microcontroller into a more accessible package.

Each of the sensor nodes are connected to the respective arduino from which the data streaming takes place. These nodes are connected to a single server raspberry pi to store and maintain the database.



ARDUINO program for each sensor

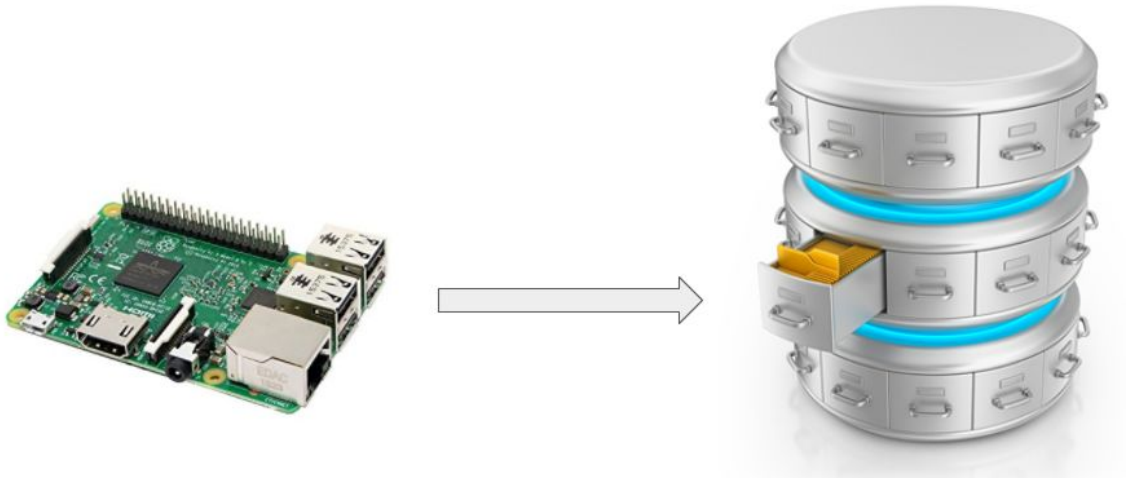
```
#include <SoftwareSerial.h>

int LED = 13;

int isObstaclePin = 7;
```

```
int isObstacle = HIGH;
SoftwareSerial bt (10,11);
void setup() {
  pinMode(LED, OUTPUT);
  pinMode(isObstaclePin, INPUT);
  bt.begin(9600);
  Serial.begin(9600);
}
void loop() {
  isObstacle = digitalRead(isObstaclePin);
  if (isObstacle == HIGH)
  {
    Serial.println("OBSTACLE!!, OBSTACLE!!");
    digitalWrite(LED, HIGH);
    bt.print (String(1));
    bt.print("\n");
    delay (200);
  }
  else
  {
    Serial.println("clear");
    digitalWrite(LED, LOW);
    bt.print (String(0));
    bt.print("\n");}
  delay(200);}
```

3.2 Network setup

The CSV database stored is then streamed into the ARIMA model explained in Chapter 2. Five different datas are injected and processed separately and being stored

back in the database to predict the future five values of individual environment variable. For example we used past days of data to predict the next day's data and this data in return is fed back to the model to predict the next day's and so on. The data is processed at this node is being updated in the database to make it ready to stream.



3.3 ARIMA implementation

The saved database is used by ARIMA to predict the future values. The predicted value are then appended on previously saved data and again ARIMA is run on that data so now ARIMA uses previously saved data along with newly generated predicted value to predict new value. On each step this process is repeated five times individually for each of the five environment variables. This process is repeated each time you want to predict any values.

The ARIMA program consists of two section ARIMA itself and GridParameter evaluator program for (p,d,q) values.Below is the program for both:

ARIMA main program

```
from statsmodels.tsa.arima_model import ARIMA
import numpy
```

```
def difference(dataset, interval=1):
diff = list()
for i in range(interval, len(dataset)):
        value = float(dataset[i]) - float(dataset[i - interval])
        diff.append(value)
return numpy.array(diff)

def inverse_difference(history, yhat, interval=1):
        return yhat + history[-interval]

def model(path):
series = Series.from_csv(path, header=None)
X = series.values
days_in_year = len(X)-24
differenced = difference(X, days_in_year)
print(differenced)
model = ARIMA(differenced, order=(3,1,0))
print(model)
model_fit = model.fit(disp=0)
start_index = len(differenced)
end_index = start_index + 23
forecast = model_fit.predict(start=start_index, end=end_index)
history = [x for x in X]
day = 1

for yhat in forecast:
        inverted = inverse_difference(history, yhat, days_in_year)
print('Day %d: %f' % (day, inverted))
history.append(inverted)
```

```
day += 1
with open('test2.csv', "w") as output:
writer = csv.writer(output, lineterminator='\n')
for val in history:
        writer.writerow([val])
model('~/Documents/BTP_WOT/arima/git/csv/humidity.csv')
```

GridParameter evaluator program

```
import math
import warnings
from pandas import read_csv,Series
from pandas import datetime
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
import numpy as np

def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

from numpy import mean, absolute

def mad(data, axis=None):
        return mean(absolute(data - mean(data, axis)), axis)

# evaluate an ARIMA model for a given order (p,d,q)
def evaluate_arima_model(X, arima_order):
# prepare training dataset
```

```
        train_size = int(len(X) * 0.66)
        train, test = X[0:train_size], X[train_size:]
        history = [x for x in train]
# make predictions
        predictions = list()
        for t in range(len(test)):
                model = ARIMA(history, order=arima_order)
                model_fit = model.fit(disp=0)
                yhat = model_fit.forecast()[0]
                predictions.append(yhat)
                history.append(test[t])
# calculate out of sample error
#print('MAPE : '+mean_absolute_percentage_error(test,predictions))
#print('MAD : '+mad(predictions))
        ma = mad(predictions)
        mape = mean_absolute_percentage_error(test,predictions)
        error = mean_squared_error(test, predictions)
        return error,ma,mape
# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values):
        dataset = dataset.astype('float32')
        best_score, best_cfg = float("inf"), None
        for p in p_values:
                for d in d_values:
                        for q in q_values:
                        order = (p,d,q)
                        try:
                        mse,ma,mape = evaluate_arima_model(dataset, order)
                        if mse < best_score:
```

```
                    best_score, best_cfg = mse, order

                    print('ARIMA%s  MSE=%.3f' % (order,mse))
                    print('\t\tMAPE=%.3f' % (mape))
                    print('\t\tRMSE=%.3f' % (math.sqrt(mse)))

                    except:

                    continue

        print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))

        return best_cfg

# load dataset

def parser(x):

        return datetime.strptime(x, '%Y-%m-%d')

series=Series.from_csv('~/Documents/BTP_WOT/arima/git/daily-minimum-temp
eratures-in-me.csv', header=None)

p_values = range(0,7)

d_values = range(0, 3)

q_values = range(0, 3)

warnings.filterwarnings("ignore")

result = evaluate_models(series.values, p_values, d_values, q_values)
```

3.4. Network and tunnel

The updated database contains the predicted and missing values in place is ready to be exported to the Second Life. The CSV hourly data is packed into JSON objects to create requests to the secondlife server. This has been implemented with python scripts which is found below.

```
import sys, getopt

import csv

import json
```

```
#Get Command Line Arguments
def main(argv):
    input_file = ''
    output_file = ''
    format = ''
    try:
        opts, args = getopt.getopt(argv,"hi:o:f:",["ifile=","ofile=","format="])
    except getopt.GetoptError:
        print 'csv_json.py -i <path to inputfile> -o <path to outputfile> -f
<dump/pretty>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'csv_json.py -i <path to inputfile> -o <path to outputfile> -f
<dump/pretty>'
            sys.exit()
        elif opt in ("-i", "--ifile"):
            input_file = arg
        elif opt in ("-o", "--ofile"):
            output_file = arg
        elif opt in ("-f", "--format"):
            format = arg
    read_csv(input_file, output_file, format)


#Read CSV File
def read_csv(file, json_file, format):
    csv_rows = []
    with open(file) as csvfile:
        reader = csv.DictReader(csvfile)
```

```
        title = reader.fieldnames
        for row in reader:
            csv_rows.extend([{title[i]:row[title[i]] for i in range(len(title))}])
        write_json(csv_rows, json_file, format)

#Convert csv data into json and write it
def write_json(data, json_file, format):
    with open(json_file, "w") as f:
        if format == "pretty":
            f.write(json.dumps(data, sort_keys=False, indent=4, separators=(',', ':
'),encoding="utf-8",ensure_ascii=False))
        else:
            f.write(json.dumps(data))

if __name__ == "__main__":
    main(sys.argv[1:])
```

3.5 SecondLife Implementation

With help of llHttprquest command we can request the data in json format from the server. We convert data from json to simple string and analyse them so that we can assign values to different parameters and graphics to change the environment inside second life accordingly. In this way journey of the data from sensoor to second life giving us the virtual environment we needed to make.

There were two scripts used for this purpose. One to is the base one to convert secondlife environment for wind speed, precipitation, temperature and visibility. Other one is for assigning direction to prims formed by llparticlesystem function through above scripts in accordance to the wind direction fetched from the server. Both program are as follows:

```
string LINK_D = "http://3aae4093.ngrok.io";
```

```
float dlay=5;


list
A12AM=["0.225","0.225","0.225","0.12","0.12","0.12","0.116","0.119","0.22",
"0.068","0.081","0.11"];

list
A6AM=["0.079","0.217","0.435","0.103","0.205","0.24","0.79","0.79","0.79","
0.27","0.154","0.21"];

list
A12PM=["0.122","0.224","0.38","0.248","0.248","0.32","0.245","0.261","0.3",
"0.35","0.35","0.35"];

list
A6PM=["0.073","0.2","0.4","0.054","0.107","0.125","0.946","0.946","0.946","
0.34","0.27","0.27"];


integer t;

float temp = 9;

float prec = 0.3;

float snow_depth = 9;

float vis = 0.1;

float acc=-0.2;

float ws;

float wd;

integer toogle=1;


rain()

{

   llParticleSystem([

                    // start of particle settings

          // Texture Parameters:
```

```
        PSYS_SRC_TEXTURE, llGetInventoryName(INVENTORY_TEXTURE,
0),


       /*  PSYS_PART_START_SCALE, <0.10,.5, FALSE>,
PSYS_PART_END_SCALE, <.05,4.0, FALSE>,*/

 PSYS_PART_START_SCALE, <0.05,0.05, FALSE>,
PSYS_PART_END_SCALE, <0.25,0.75, FALSE>,


        PSYS_PART_START_COLOR, <0.498, 0.859, 1.000>,
PSYS_PART_END_COLOR, <0.000, 0.455, 0.851>,


        PSYS_PART_START_ALPHA, (float).7*toogle,
PSYS_PART_END_ALPHA, (float).5*toogle,




        // Production Parameters:


        PSYS_SRC_BURST_PART_COUNT, (integer)90000*100*100,


        PSYS_SRC_BURST_RATE, (float) 1,


        PSYS_PART_MAX_AGE, (float)30.0,


        PSYS_SRC_MAX_AGE,(float) 0.0,
```

```
        // Placement Parameters:


        PSYS_SRC_PATTERN, (integer)8, // 1=DROP, 2=EXPLODE, 4=ANGLE,
8=ANGLE_CONE,




        // Placement Parameters (for any non-DROP pattern):


        PSYS_SRC_BURST_SPEED_MIN, (float)0.0,
PSYS_SRC_BURST_SPEED_MAX, (float)0.2,


        PSYS_SRC_BURST_RADIUS, 25.0,




        // Placement Parameters (only for ANGLE & CONE patterns):

        PSYS_SRC_ANGLE_BEGIN, (float) .33*PI,   PSYS_SRC_ANGLE_END,
(float)2*PI,


      // PSYS_SRC_OMEGA, <0,0,0>,




        // After-Effect & Influence Parameters:


        PSYS_SRC_ACCEL, <0.0,0.0,-0.2>,
```

```
        // PSYS_SRC_TARGET_KEY,
(key)"0ef87053-6c78-7a04-df64-4e2827ec51b1",



        PSYS_PART_FLAGS, (integer)( 0         // Texture Options:

                            | PSYS_PART_INTERP_COLOR_MASK

                            | PSYS_PART_INTERP_SCALE_MASK

                            | PSYS_PART_EMISSIVE_MASK

                        // | PSYS_PART_FOLLOW_VELOCITY_MASK

                                    // After-effect & Influence Options:

                        //   | PSYS_PART_WIND_MASK

                         //  | PSYS_PART_BOUNCE_MASK

                        // | PSYS_PART_FOLLOW_SRC_MASK

                        // | PSYS_PART_TARGET_POS_MASK

                        // | PSYS_PART_TARGET_LINEAR_MASK
```

```
                    )


        //end of particle settings

                ]);
}


snow()
{   llOwnerSay((string)toogle+"**");
   llParticleSystem([



                    PSYS_SRC_TEXTURE,
llGetInventoryName(INVENTORY_TEXTURE, 0),
                    PSYS_PART_START_SCALE, <.4,.4, FALSE>,
                    PSYS_PART_END_SCALE, <.2,.2, FALSE>,
                    PSYS_PART_START_COLOR, <1,1,1>,
                    PSYS_PART_END_COLOR, <1,1,1>,
                    PSYS_PART_START_ALPHA, (float).9*toogle,
                    PSYS_PART_END_ALPHA, (float).1*toogle,


                    PSYS_PART_MAX_AGE, 10500.0,
                    PSYS_SRC_MAX_AGE, 0.0,
                    PSYS_SRC_BURST_RATE, 0.1,
                    PSYS_SRC_BURST_PART_COUNT, 900*100,


                    // Placement Parameters:
        PSYS_SRC_PATTERN, (integer)8, // 1=DROP, 2=EXPLODE, 4=ANGLE,
8=ANGLE_CONE,
```

```
        // Placement Parameters (for any non-DROP pattern):

        PSYS_SRC_BURST_SPEED_MIN, (float).0,
PSYS_SRC_BURST_SPEED_MAX, (float).2,

        PSYS_SRC_BURST_RADIUS, 25.0,


        // Placement Parameters (only for ANGLE & CONE patterns):

        PSYS_SRC_ANGLE_BEGIN, (float) .33*PI,   PSYS_SRC_ANGLE_END,
(float)2*PI,

      // PSYS_SRC_OMEGA, <0,0,0>,


        // After-Effect & Influence Parameters:

        PSYS_SRC_ACCEL, <.0,.0,acc>,

         PSYS_SRC_TARGET_KEY,
(key)"0ef87053-6c78-7a04-df64-4e2827ec51b1",


        PSYS_PART_FLAGS, (integer)( 0        // Texture Options:

                    | PSYS_PART_INTERP_COLOR_MASK

                    | PSYS_PART_INTERP_SCALE_MASK

                    | PSYS_PART_EMISSIVE_MASK

                   // | PSYS_PART_FOLLOW_VELOCITY_MASK

                                // After-effect & Influence Options:

                   //   | PSYS_PART_WIND_MASK

                   // | PSYS_PART_BOUNCE_MASK

                   // | PSYS_PART_FOLLOW_SRC_MASK

                     | PSYS_PART_TARGET_POS_MASK

                   // | PSYS_PART_TARGET_LINEAR_MASK

                  )
```

```
                ]);
}

list region(){

    if(t>=0 && t<=6)
        return change_day(A12AM,A6AM,0);
    else if(t>=6 && t<=12)
         return change_day(A6AM,A12PM,6);
    else if(t>=12 && t<=18)
        return change_day(A12PM,A6PM,12);
    else return change_day(A6PM,A12AM,18);
}

list change_day(list str,list end,integer to){

    list new;
    float d;
    float val;
    integer i;
    for(i=0;i<12;i++){

        d=(float)llList2String(end,i)-(float)llList2String(str,i);
        d=d/6;
        val= d*(t-to) + (float)llList2String(str,i);

        new += (string)val;
```

```
        // llOwnerSay("+"+(string)val+"+");
        // llOwnerSay(""+llList2String(new,i)+"");
    }

    t=(t/24);
    llOwnerSay("@setenv_sunmoonposition:"+ (string)t  +"=force");
    return new;
}

list temperature(list arr,float t)
{
    float d;
    float val;
    d = (0.2/28)*(16-t);
    float valr = (float)llList2String(arr,6)-d;
    float valg = (float)llList2String(arr,7)-d;
    float valb = (float)llList2String(arr,8)-d;
    arr =llListReplaceList(arr,[(string)valr],6,6);
    arr =llListReplaceList(arr,[(string)valg],7,7);
    arr =llListReplaceList(arr,[(string)valb],8,8);
    return arr;
}
Precipitation(float prec)
{
  if((string)prec == "T" || (prec < 0.5 && prec >= 0)){
    snow();}
  else if(prec > 0.5)
```

```
    rain();
}
Visibility(float vis)
{
    llOwnerSay("@setenv_densitymultiplier:"+(string)0.0909+"=force");
  vis=vis/16;
  vis=vis*1000;
  llOwnerSay("@setenv_distancemultiplier:"+(string)vis+"=force");
}
default
{
    state_entry() {
        llSay(0, "Hello world.");
        llHTTPRequest(LINK_D, [], "");
        llSetTimerEvent(dlay);

    }
    timer(){
        llHTTPRequest(LINK_D, [], "");
    }

    http_response(key request_id, integer status, list metadata, string body) {

        string str;
        integer num;
        if (status == 200) {
```

```
        list
row=["temperature","visibility","precipitation","windspeed","winddirection","
time"];



        str=llJsonGetValue(body,[llList2String(row,0)]);llOwnerSay(str);
        temp=(float)str;
        str=llJsonGetValue(body,[llList2String(row,1)]);llOwnerSay(str);
        vis=(float)str;//llOwnerSay((string)vis);
        str=llJsonGetValue(body,[llList2String(row,2)]);llOwnerSay(str);
        prec=(float)str;
        str=llJsonGetValue(body,[llList2String(row,3)]);llOwnerSay(str);
        ws=(float)str;
        str=llJsonGetValue(body,[llList2String(row,4)]);llOwnerSay(str);
        wd=(float)str;
        str=llJsonGetValue(body,[llList2String(row,5)]);llOwnerSay(str);
        t=(integer)str;



    acc=(-1)*((ws-3)*(0.15));


    list new;
   // llOwnerSay("YAY!");
    new = region();
    new = temperature(new,temp);
    toogle=1;
    if(prec==0)
     toogle=0;
```

```
        Precipitation(prec);
        Visibility(vis);
        llOwnerSay("@setenv_bluedensityr:"+ llList2String(new,0) +"=force");
        llOwnerSay("@setenv_bluedensityg:"+ llList2String(new,1) +"=force");
        llOwnerSay("@setenv_bluedensityb:"+ llList2String(new,2) +"=force");
        llOwnerSay("@setenv_bluehorizonr:"+ llList2String(new,3) +"=force");
        llOwnerSay("@setenv_bluehorizong:"+ llList2String(new,4) +"=force");
        llOwnerSay("@setenv_bluehorizonb:"+ llList2String(new,5) +"=force");
        llOwnerSay("@setenv_sunmooncolorr:"+ llList2String(new,6) +"=force");
        llOwnerSay("@setenv_sunmooncolorg:"+ llList2String(new,7) +"=force");
        llOwnerSay("@setenv_sunmooncolorb:"+ llList2String(new,8) +"=force");
        llOwnerSay("@setenv_ambientr:"+ llList2String(new,9) +"=force");
        llOwnerSay("@setenv_ambientg:"+ llList2String(new,10) +"=force");
        llOwnerSay("@setenv_ambientb:"+ llList2String(new,11) +"=force");

        }
        else {
            llOwnerSay("Unable to fetch "+LINK_D);
        }
    }
    }
```

for wind direction

```
string LINK_D = "http://3aae4093.ngrok.io";
float dlay=5;
default
{
```

```
state_entry() {
    llSay(0, "Hello world.");
    llOwnerSay(llGetKey());
    llHTTPRequest(LINK_D, [], "");;
    llSetTimerEvent(dlay);

}
timer(){
    llHTTPRequest(LINK_D, [], "");
}

http_response(key request_id, integer status, list metadata, string body) {

    string str;
    integer num;
    string wd;
    float ws;
    if (status == 200) {

        list
row=["temperature","visibility","precipitation","windspeed","winddirection","
time"];

        str=llJsonGetValue(body,[llList2String(row,4)]);//llOwnerSay(str);
        wd=str;
       // str=llJsonGetValue(body,[llList2String(row,4)]);//llOwnerSay(str);
        //ws=(float)str;
```

```
        list
dir=["North","NNE","NE","NEE","East","ESE","SE","SSE","South","SSW
","SW","WSW","West","WNW","NW","NNW","Calm","Variable"];


        float x;float y;float theta;
        integer i;
        if(str=="Calm"){
           x=0;y=0;
        }
        else if(str=="Variable"){

           x=llFrand(50)-25;
           y=llFrand(50)-25;
           }
           else{

           for(i=0;i<16;i++){

              if(str==llList2String(dir,i)){

                 theta=((float)i)*22.5;
                 theta+=85;
                 x=25*llSin(theta);
                 y=25*llCos(theta);
                 i=16;
              }
           }
        }
```

```
            llOwnerSay((string)(x)+" "+(string)(y)+" "+str);
            llSetPos(llGetPos() + <x,y,0>);
            llSleep(dlay+10);
            llSetPos(llGetPos() + <-x,-y,0>);
        }
        else {
            llOwnerSay("Unable to fetch "+LINK_D);
        }

    }

}
```

Chapter 4

Future Scope

Extreme Learning Machine is a promising straightforward algorithm which can achieve a relatively high Overall Accuracy and can significantly decrease the time of the training phase. The models that we built here works only for this type of weather data and is confined to this scope. Through this project we realise that as the number of parameters increase the closeness of the environment increases to mimic the similarity. Through this project we also realised the capabilities and limitations of FLASK server hosting as well as database handling.

While implementing the model we came across roadblocks for trivial values of P,D,Q as mentioned in Chapter 2. For future research, perhaps a better dataset values that could not just determine the weather condition but also other factors of the environment including the avator characteristics would improve the quality and motivation on Secondlife.

References

[1] http://wiki.secondlife.com/wiki/LlParticleSystem

[2] https://www.arduino.cc/en/Main/Docs

[3] http://flask.pocoo.org/docs/0.12/

[4] https://ngrok.com/

[5]https://www.researchgate.net/publication/285356304_A_Meta-Synthesis_of_Research_Using_Multiuser_Virtual_Environments_ie_Second_Life_in_Teacher_Education

[6] http://www.restapitutorial.com/lessons/whatisrest.html

[7] https://machinelearningmastery.com/make-sample-forecasts-arima-python/

[8] https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3

[9] https://www.slideshare.net/21_venkat/arima-26196965

[10] http://siteresources.worldbank.org/DEC/Resources/Hevia_ARMA_estimation

http://siteresources.worldbank.org/DEC/Resources/Hevia_ARMA_estimation.pdf

[11] http://wiki.secondlife.com/wiki/LSL_Protocol/RestrainedLoveAPI

[12] http://www.firestormviewer.org/

[13] http://www.statsmodels.org/stable/index.html

[14] http://www.numpy.org/

[15] https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

Appendix A

llParticleSystem Parameter Sheet

A description of all the parameter that can be passed to llparticlesystem inside secondlife script.

Rule / Value Constant	Rule Parameter	Description	Value	
System Behavior				
PSYS_PART_FLAGS		integer flags	Various flags controlling the behavior of the particle system. The value may be specified as an integer in decimal or hex format, or by ORing together (using the | operator) one or more of the following flag constants:	0
Values	PSYS_PART_BOUNCE_MASK		When set, specifies particles will bounce off a plane at the region Z height of the emitter. On "bounce", each particle reverses velocity and angle. This only works for particles above the plane falling down on it.	0x004
	PSYS_PART_EMISSIVE_MASK		When set, particles are full-bright and are unaffected by global lighting (sunlight). Otherwise, particles will be lit depending on the current global lighting conditions. Note that point lights **do** illuminate non-emissive particles.	0x100
	PSYS_PART_FOLLOW_SRC_MASK		When set, particles move relative to the position of the emitter. Otherwise, particle position and movement are unaffected by the position/movement of the emitter.	0x010

		This flag disables the PSYS_SRC_BURST_RADIUS rule.	
	PSYS_PART_FOLLOW_VELOCITY_MASK	When set, particles rotate to orient their "top" towards the direction of movement or emission. Otherwise, particles are oriented vertically as their textures would appear (top of texture at top, left at left).	0x020
	PSYS_PART_INTERP_COLOR_MASK	When set, particle color and alpha transition from their START settings to their END settings during the particle's lifetime. The transition is a smooth interpolation.	0x001
	PSYS_PART_INTERP_SCALE_MASK	When set, particle size/scale transitions from its START setting to its END setting during the particle's lifetime.	0x002
	PSYS_PART_RIBBON_MASK	Joins a stream of particles together into a continuous strip. Particle textures are stretched (or squeezed) to join their right edges to their predecessor's left. Ribbon 'width' is controlled by the 'x' values of start and end scale. (The 'y' values are ignored. The distance between particles controls the 'length' of each ribbon segment). Unlike other particle effects, ribbon segments are not rendered facing the viewer's camera. The Z axis of each new particle mimics the Z axis of the emitter prim. Ribbon segments will not render if they have no 'length' - this happens when particles move only 'up' or 'down' the local Z-axis of their emitter prim. PSYS_PART_FOLLOW_VELOCITY_MASK has no effect on ribbons. For a simple ribbon effect, try using the DROP pattern, TEXTURE_BLANK, ACCEL and/or WIND.	0x400
	PSYS_PART_TARGET_LINEAR_MASK	When set, emitted particles move in a straight evenly-spaced line towards the target specified by the PSYS_SRC_TARGET_KEY rule. This option ignores all Non-DROP patterns and their dependent attributes (radius, burst speeds, angles, and omega). PSYS_SRC_ACCEL and	0x080

		PSYS_PART_WIND_MASK are ignored as well. Using PSYS_PART_BOUNCE_MASK while the target is 'below' the emitter will cause the linear particle stream to deflect upwards, terminating above the target.	
	PSYS_PART_TARGET_POS_MASK	When set, emitted particles change course during their lifetime, attempting to move towards the target specified by the PSYS_SRC_TARGET_KEY rule by the time they expire. Note that if no target is specified, the target moves out of range, or an invalid target is specified, the particles target the prim itself.	0x040
	PSYS_PART_WIND_MASK	When set, particle movement is affected by the wind. It is applied as a secondary force on the particles.	0x008
	PSYS_PART_BEAM_MASK	(*unimplemented*) mask but in the enum	0x200
	LL_PART_HUD	Used by the viewer to keep HUD and World particle sources separate.	0x40000000
	LL_PART_DEAD_MASK	Removes particles, not compatible with any other PSYS_PART_*_MASK	0x80000000
System Presentation			
PSYS_SRC_PATTERN	integer pattern	Specifies the general emission pattern.	9
Values	PSYS_SRC_PATTERN_EXPLODE	Sprays particles outwards in a spherical area. The Initial velocity of each particle is determined by PSYS_SRC_BURST_SPEED_MIN and PSYS_SRC_BURST_SPEED_MAX. The EXPLODE pattern ignores the ANGLE parameters.	0x02
	PSYS_SRC_PATTERN_ANGLE_CONE	Sprays particles outwards in a spherical, sub-spherical, conical or ring shaped area, as defined by the ANGLE parameters PSYS_SRC_ANGLE_BEGIN and PSYS_SRC_ANGLE_END. The ANGLE_CONE pattern can be used	0x08

		to imitate the EXPLODE pattern by explicitly setting PSYS_SRC_ANGLE_BEGIN to 0.00000 and PSYS_SRC_ANGLE_END to 3.14159 (or PI) (or vice versa).	
	PSYS_SRC_PATTERN_ANGLE	Sprays particles outward in a flat circular, semi-circular, arc or ray shaped areas, as defined by PSYS_SRC_ANGLE_BEGIN and PSYS_SRC_ANGLE_END. The circular pattern radiates outwards around the prim's local X axis line.	0x04
	PSYS_SRC_PATTERN_DROP	Creates particles with no initial velocity. The DROP pattern will override any values given for PSYS_SRC_BURST_RADIUS, PSYS_SRC_BURST_SPEED_MIN, and PSYS_SRC_BURST_SPEED_MAX, setting each to 0.00000. (All patterns will behave like the DROP pattern, if RADIUS, SPEED_MIN and SPEED_MAX are explicitly set to 0.0000.)	0x01
	PSYS_SRC_PATTERN_ANGLE_CONE_EMPTY	(*incomplete implementation*) acts the same as the PSYS_SRC_PATTERN_DROP pattern, it is believed that the original intention for this pattern was to invert the effect of the ANGLE parameters, making them delineate an area where particles were NOT to be sprayed. (effectively the inverse or opposite of the behavior of the ANGLE_CONE pattern).	0x10
PSYS_SRC_BURST_RADIUS	float radius	Specifies the distance from the emitter where particles will be created. This rule is ignored when the PSYS_PART_FOLLOW_SRC_MASK flag is set. A test in http://forums-archive.secondlife.com/327/f5/226722/1.html indicates that the maximum value is 50.00	16

PSYS_SRC_ANGLE_BEGIN	float angle_begin	Specifies a half angle, in radians, of a circular or spherical "dimple" or conic section (starting from the emitter facing) within which particles will NOT be emitted. Valid values are the same as for PSYS_SRC_ANGLE_END, though the effects are reversed accordingly. If the pattern is PSYS_SRC_PATTERN_ANGLE, the presentation is a 2D flat circular section. If PSYS_SRC_PATTERN_ANGLE_CONE or PSYS_SRC_PATTERN_ANGLE_CONE_EMPTY is used, the presentation is a 3D spherical section. Note that the value of this parameter and PSYS_SRC_ANGLE_END are internally re-ordered such that this parameter gets the smaller of the two values.	22
PSYS_SRC_ANGLE_END	float angle_end	Specifies a half angle, in radians, of a circular or spherical "dimple" or conic section (starting from the emitter facing) within which particles WILL be emitted. Valid values are 0.0, which will result in particles being emitted in a straight line in the direction of the emitter facing, to PI, which will result in particles being emitted in a full circular or spherical arc around the emitter, not including the "dimple" or conic section defined by PSYS_SRC_ANGLE_BEGIN. If the pattern is PSYS_SRC_PATTERN_ANGLE, the presentation is a 2D flat circular section. If PSYS_SRC_PATTERN_ANGLE_CONE or PSYS_SRC_PATTERN_ANGLE_CONE_EMPTY is used, the presentation is a 3D spherical section. Note that the value of this parameter and PSYS_SRC_ANGLE_BEGIN are internally re-ordered such that this parameter gets the larger of the two values.	23
PSYS_SRC_INNERANGLE	float angle_inner	**DEPRECATED: Use PSYS_SRC_ANGLE_BEGIN instead.** Works similar to its replacement rule, except the edge of the section is aligned with the	10

		emitter facing, rather than its center.	
PSYS_SRC_OUTERANGLE	float angle_outer	**DEPRECATED: Use PSYS_SRC_ANGLE_END instead.** Works similar to its replacement rule, except the edge of the section is aligned with the emitter facing, rather than the section's center.	11
PSYS_SRC_TARGET_KEY	key target	Specifies the key of a target object, prim, or agent towards which the particles will change course and move (if PSYS_PART_TARGET_POS_MASK is specified) or will move in a straight line (if PSYS_PART_TARGET_LINEAR_MASK is specified). They will attempt to end up at the geometric center of the target at the end of their lifetime. Requires the PSYS_PART_TARGET_POS_MASK or PSYS_PART_TARGET_LINEAR_MASK flag be set. caveat 4	20
Particle Appearance			
PSYS_PART_START_COLOR	vector color_start	A vector specifying the color of the particles upon emission.	1
PSYS_PART_END_COLOR	vector color_end	A vector specifying the color the particles transition to during their lifetime. Only used if the PSYS_PART_INTERP_COLOR_MASK flag is set.	3
PSYS_PART_START_ALPHA	float alpha_start	Specifies the alpha of the particles upon emission. Valid values are in the range 0.0 to 1.0. Lower values are more transparent; higher ones are more opaque.	2
PSYS_PART_END_ALPHA	float alpha_end	Specifies the alpha the particles transition to during their lifetime. Only used if the PSYS_PART_INTERP_COLOR_MASK flag is set. Valid values are the same as PSYS_PART_START_ALPHA.	4

PSYS_PART_START_SCALE	vector scale_start	Specifies the scale or size of the particles upon emission. Valid values for each direction are 0.03125 to 4.0, in meters. The actual particle size is always a multiple of 0.03125. Smaller changes don't have any effect. Since particles are essentially 2D sprites, the Z component of the vector is ignored and can be set to 0.0.	5
PSYS_PART_END_SCALE	vector scale_end	Specifies the scale or size the particles transition to during their lifetime. Only used if the PSYS_PART_INTERP_SCALE_MASK flag is set. Valid values are the same as PSYS_PART_START_SCALE.	6
PSYS_SRC_TEXTURE	string texture	Specifies the name of a texture in the emitter prim's inventory to use for each particle. Alternatively, you may specify an asset key UUID for a texture. If using llLinkParticleSystem and texture is not a UUID, texture must be in the emitter prim (not necessarily with the script).	12
PSYS_PART_START_GLOW	float glow_start	Specifies the glow of the particles upon emission. Valid values are in the range of 0.0 (no glow) to 1.0 (full glow).	26
PSYS_PART_END_GLOW	float glow_end	Specifies the glow that the particles transition to during their lifetime. Valid values are the same as PSYS_PART_START_GLOW.	27
Particle Blending			
Note: The particle system blend parameters wrap directly to OpenGL's glBlendFunc. Detailed documentation for glBlendFunc, including formulas, can be found in the official glBlendFunc documentation			
PSYS_PART_BLEND_FUNC_SOURCE	integer bf_source	Specifies how blending function uses the incoming particle's color and alpha information to produce the rendered result. Defaults to PSYS_PART_BF_SOURCE_ALPHA.	24

PSYS_PART_BLEND_FUNC_DEST	integer bf_dest	Specifies how blending function uses the current framebuffer's color and alpha information to produce the rendered result. Defaults to PSYS_PART_BF_ONE_MINUS_SOURCE_ALPHA. To make particles blend with the background in a less opaque and more luminescent way use PSYS_PART_BF_ONE for dest and the default for source. Most other blending combinations will render the invisible/alpha portion of your particle texture, unless the invisible area of your texture is all black (or, in some cases, unless it is all white).	25
Values	PSYS_PART_BF_ONE	Do not scale the source or destination RGBA values.	0x0
	PSYS_PART_BF_ZERO	Zero out the source or destination RGBA values.	0x1
	PSYS_PART_BF_DEST_COLOR	Scale the RGBA values by the RGBA values of the destination.	0x2
	PSYS_PART_BF_SOURCE_COLOR	Scale the RGBA values by the RGBA values of the particle source.	0x3
	PSYS_PART_BF_ONE_MINUS_DEST_COLOR	Scale the RGBA values by the inverted RGBA values of the destination.	0x4
	PSYS_PART_BF_ONE_MINUS_SOURCE_COLOR	Scale the RGBA values by the inverted RGBA values of the particle source.	0x5
	PSYS_PART_BF_SOURCE_ALPHA	Scale the RGBA values by the alpha values of the particle source.	0x7
	PSYS_PART_BF_ONE_MINUS_SOURCE_ALPHA	Scale the RGBA values by the inverted alpha values of the particle source.	0x9
Particle Flow			

PSYS_SRC_MAX_AGE	float duration_system	Specifies the length of time, in seconds, that the emitter will operate upon coming into view range (if the particle system is already set) or upon execution of this function (if already in view range). Upon expiration, no more particles will be emitted, except as specified above. Zero will give the particle system an infinite duration. (caveat 1)	19
PSYS_PART_MAX_AGE	float duration_particle	Specifies the lifetime of each particle emitted, in seconds. Maximum is 30.0 seconds. During this time, the particle will appear, change appearance and move according to the parameters specified in the other sections, and then disappear.	7
PSYS_SRC_BURST_RATE	float burst_sleep	Specifies the time interval, in seconds, between "bursts" of particles being emitted. Specifying a value of 0.0 will cause the emission of particles as fast as the viewer can do so.	13
PSYS_SRC_BURST_PART_COUNT	integerburst_particle_count	Specifies the number of particles emitted in each "burst".	15
Particle Motion			
PSYS_SRC_ACCEL	vector acceleration	Specifies a directional acceleration vector applied to each particle as it is emitted, in meters per second squared. Valid values are 0.0 to 100.0 for each direction both positive and negative, as region coordinates.	8
PSYS_SRC_OMEGA	vector omega	Sets how far to rotate the "pattern" after each particle burst. (Burst frequency is set with PSYS_SRC_BURST_RATE.) Omega values are approximately 'radians per burst' around the prim's global (not local) X,Y,Z axes. For precise and predictable pattern rotation, rotate the prim instead of using PSYS_SRC_OMEGA. Omega has no visible effect on drop, explode and certain specific angle and angle cone patterns, depending on prim orientation. Pattern rotation can be	21

		used with prim orientation and llTargetOmega() but won't produce consistent results. (caveat 2 and caveat 3)	
PSYS_SRC_BURST_SPEED_MIN	float speed_min	Specifies the minimum value of a random range of values which is selected for each particle in a burst as its initial speed upon emission, in meters per second. Note that the value of this parameter and PSYS_SRC_BURST_SPEED_MAX are internally re-ordered such that this parameter gets the smaller of the two values.	17
PSYS_SRC_BURST_SPEED_MAX	float speed_max	Specifies the maximum value of a random range of values which is selected for each particle in a burst as its initial speed upon emission, in meters per second. Note that the value of this parameter and PSYS_SRC_BURST_SPEED_MIN are internally re-ordered such that this parameter gets the larger of the two values.	18

Appendix B

A description of all the syntax or command that can be used under the RestrainedLoveAPI library.

Viewer Control

- ***Allow/prevent changing some debug settings*** : @setdebug=<y/n>

Implemented in v1.16

When prevented, the user is unable to change some viewer debug settings (Advanced > Debug Settings). As most debug settings are either useless or critical to the user's experience, a whitelist approach is taken : only a few debug settings are locked, the others are always available and untouched.

- ***Force change a debug setting*** : @setdebug_<setting>:<value>=force (*)

Implemented in v1.16

Forces the viewer to change a particular debug setting and set it to <value>. This command is actually a package of many sub-commands, that are regrouped under "@setdebug_...", for instance "@setdebug_avatarsex:0=force", "@setdebug_renderresolutiondivisor:64=force" etc.

The debug settings that can be changed are :

- AvatarSex (0 : Female, 1 : Male) : gender of the avatar at creation.
- RenderResolutionDivisor (1 -> ...) : "blurriness" of the screen. Combined to clever @setenv commands, can simulate nice effects. Note: renderresolutiondivisor is a Windlight only option (Basic Shaders must be enabled in graphics preferences) and as such, is not available in v1.19.0.5 or older viewers.

- ***Get the value of a debug setting*** : @getdebug_<setting>=<channel_number>

Implemented in v1.16

Makes the viewer automatically answer the value of a debug setting, immediately on the chat channel number <channel_number> that the script can listen to. Always use

a positive integer. Remember that regular viewers do not answer anything at all so remove the listener after a timeout. The answer is the value that has been set with the <setting> part of the matching @setdebug command, or by hand.

The debug settings that can be changed are :

- AvatarSex (0 : Female, 1 : Male) : gender of the avatar at creation.
- RenderResolutionDivisor (1 -> ...) : "blurriness" of the screen. Combined to clever @setenv commands, can simulate nice effects. Note: renderresolutiondivisor is a Windlight only option (Basic Shaders must be enabled in graphics preferences) and as such, is not available in v1.19.0.5 or older viewers.
- RestrainedLoveForbidGiveToRLV (0/1) : When set to 1, the RLV does not put temporary folders (folders which name begins with '~') directly into the "#RLV" folder, but under "Inventory" instead.
- RestrainedLoveNoSetEnv (0/1) : When set to 1, @setenv commands are ignored.
- WindLightUseAtmosShaders (0/1) : When set to 1, Windlight atmospheric shaders are enabled.

- ***Allow/prevent changing the environment settings*** : @setenv=<y/n>

Implemented in v1.14

When prevented, the user is unable to change the viewer environment settings (World > Environment Settings > Sunrise/Midday/Sunset/Midnight/Revert to region default/Environment editor are all locked out).

- ***Force change an environment setting*** : @setenv_<setting>:<value>=force (*)

Implemented in v1.14

Forces the viewer to change a particular environment setting (time of day or Windlight) and set it to <value>. This command is actually a package of many sub-commands, that are regrouped under "@setenv_...", for instance "@setenv_daytime:0.5=force", "@setenv_bluehorizonr:0.21=force" etc.

This command (like any other "force" command) is silently discarded if the corresponding restriction has been set, here "@setenv", but in this case the restriction is ignored if the change is issued from the object that has created it. In other words, a collar can restrict environment changes, yet force a change by itself, while another object could not do it until the collar lifts the restriction.

Although a range is specified for every value, no check is made in the viewer so a script can do what the UI can't do, for interesting effects. Use at your own risk, though. The ranges indicated here are merely the ones available on the sliders on the Environment Editor, for reference.

Each particular sub-command works as follows (the names are chosen to be as close to the Windlight panels of the viewer as possible) :

@setenv_XXX:<value>=force where XXX is...	**<value> range**	**Sets...**
daytime	0.0-1.0 and <0	Time of day (sunrise:0.25, midday:0.567, sunset:0.75, midnight:0.0, set back to region default:<0). **Attention, resets all other Windlight parameters**
preset	String	A Preset environment, e.g. Gelatto, Foggy. **Attention, loading a Preset is heavy on the viewer and can slow it down for a short while, don't do it every second**
ambientr	0.0-1.0	Ambient light, Red channel
ambientg	0.0-1.0	Ambient light, Green channel
ambientb	0.0-1.0	Ambient light, Blue channel
ambienti	0.0-1.0	Ambient light, Intensity
bluedensityr	0.0-1.0	Blue Density, Red channel
bluedensityg	0.0-1.0	Blue Density, Green channel
bluedensityb	0.0-1.0	Blue Density, Blue channel
bluedensityi	0.0-1.0	Blue Density, Intensity
bluehorizonr	0.0-1.0	Blue Horizon, Red channel
bluehorizong	0.0-1.0	Blue Horizon, Green channel
bluehorizonb	0.0-1.0	Blue Horizon, Blue channel
bluehorizoni	0.0-1.0	Blue Horizon, Intensity
cloudcolorr	0.0-1.0	Cloud color, Red channel

cloudcolorg	0.0-1.0	Cloud color, Green channel
cloudcolorb	0.0-1.0	Cloud color, Blue channel
cloudcolori	0.0-1.0	Cloud color, Intensity
cloudcoverage	0.0-1.0	Cloud coverage
cloudx	0.0-1.0	Cloud offset X
cloudy	0.0-1.0	Cloud offset Y
cloudd	0.0-1.0	Cloud density
clouddetailx	0.0-1.0	Cloud detail X
clouddetaily	0.0-1.0	Cloud detail Y
clouddetaild	0.0-1.0	Cloud detail density
cloudscale	0.0-1.0	Cloud scale
cloudscrollx	0.0-1.0	Cloud scroll X
cloudscrolly	0.0-1.0	Cloud scroll Y
densitymultiplier	0.0-0.9	Density multiplier of the fog
distancemultiplier	0.0-100.0	Distance multiplier of the fog
eastangle	0.0-1.0	Position of the east, 0.0 is normal
hazedensity	0.0-1.0	Density of the haze
hazehorizon	0.0-1.0	Haze at the horizon
maxaltitude	0.0-4000.0	Maximum altitude of the fog
scenegamma	0.0-10.0	Overall gamma, 1.0 is normal
starbrightness	0.0-2.0	Brightness of the stars
sunglowfocus	0.0-0.5	Focus of the glow of the sun

sunglowsize	1.0-2.0	Size of the glow of the sun
sunmooncolorr	0.0-1.0	Sun and moon, Red channel
sunmooncolorg	0.0-1.0	Sun and moon, Green channel
sunmooncolorb	0.0-1.0	Sun and moon, Blue channel
sunmooncolori	0.0-1.0	Sun and moon, Intensity
sunmoonposition	0.0-1.0	Position of the sun/moon, different from "daytime", **use this to set the apparent sunlight after loading a Preset**

Note: from the above settings, only the "daytime" one is supported by v1.19.0 (or older) viewers implementing RestrainedLove v1.14 and later. The other settings are ignored. This is because these viewers do not implement the Windlight renderer.

- ***Get the value of an environment setting*** :
  @getenv_<setting>=<channel_number>

Implemented in v1.15

Makes the viewer automatically answer the value of an environment setting, immediately on the chat channel number <channel_number> that the script can listen to. Always use a positive integer. Remember that regular viewers do not answer anything at all so remove the listener after a timeout. The answer is the value that has been set with the <setting> part of the matching @setenv command, or by hand. See the table hereabove for a list of settings.

Note: only @getenv_daytime is supported by v1.19.0 (or older, i.e. non Windlight) viewers implementing RestrainedLove v1.15 and later.