CompLACS Helicopters Scenarios

PRELIMINARY VERSION

December 12, 2012

Abstract

This document describes the three types of application scenarios developed at UCL to devise and test algorithms for our flock autonomous helicopters, one of the three application platforms of the CompLACS project.

This report is divided into three parts one for each of the three main scenarios. Each part gives a formal description of the scenario in terms of its setup, its objective, and its variations. Additionally for each of the scenario we provide some example code to aid the development of learning algorithms. Technical details specific to the scenarios implementation are reported in the appendix.

All tasks are simulated using the QRSim simulator therefore a companion to this document is the QRSim manual (`http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf`) which describes in detail the simulator and its API.

Contents

Chapter 1

Scenario 1: Cats and Mouse game

N helicopters (cats) have to be controlled in order to catch (i.e. get close to) another helicopter (mouse) at the end of the allotted time.

1.1 MDP

The underlying system is modelled as a discrete time, finite horizon MDP on a continuous state space. The system is updated at a frequency of 1Hz[1].

States: The state $s_t = (x_t^1, ..., x_t^N, x_t^m)$ at time t comprises the position ($[p_x, p_y, p_z]^\intercal$), velocity and orientation of the cats (superscripts $1, ..., N$) and of the mouse (superscript m). All are continuous variables (see section 1.3), the environment is 3 dimensional although the helicopters are assumed to fly at a fixed altitude.

Initial State: At the beginning of the task the mouse is placed at the center of the flight space[2] and the cats are positioned randomly around the mouse[3]. Initially all the helicopters are stationary (i.e. their velocities are zero).

Actions: The action $a_t = (a_t^1, ..., a_t^N)$ at time t comprises the action for each of the cats helicopters. To limit the space of control inputs we do not directly control the four inputs of each quadrotor (i.e. angles and throttle), instead each UAV is equipped with a PID controller[4] that accepts 2D linear velocity commands while maintaining a constant altitude and heading. The linear velocity commands are expressed in global coordinates (see section 1.3 for details).

[1] The update rate is user-configurable with default value of 1Hz.

[2] Without loss of generality since the control problem depends on the relative positions of mouse and cats as long as the flying area is sufficiently large.

[3] The initial position of cats are generated as:

$$\begin{cases} p_x^i = d_i \cos(\alpha_i) \\ p_y^i = d_i \sin(\alpha_i) \\ p_z^i = -h_{fix} \end{cases} \quad i \in 1..N$$

where $\alpha_i \in \mathcal{U}(0, 2\pi)$ and $d_i \in \mathcal{U}(D_m, D_M)$. The minimum and maximum distance from the mouse (D_m, D_M) and the altitude h_{fix} are user-configurable. An additional parameter D_{c2c} allows to define a minimum distance between two cats, which prevent the occurrence of an initial state in which two cats are too close.

[4] In addition to reducing the number of control inputs this makes the task closer to what is possible to implement and test using real quadrotors.

Dynamics: Markovian transition dynamics are defined by a distribution $P(s'|s,a)$ which denotes the conditional density[5] of state s' at time $t+1$ given state-action $(s_t, a_t) = (s, a)$ at time t. As the task starts the mouse helicopter tries to escape the cats by moving at a constant[6] (max) speed and following a predefined control law that prioritize escaping from the closest cats.[7]

Rewards: The task final reward is computed as the sum of the squared (2D) distances[8] of the cats to the mouse at the end of the allotted time (T):

$$r_T = -\sum_{1}^{N} d_{2D}(x_T^i, x_T^m)^2.$$

A large negative reward[9] is returned if any of the helicopters (including the mouse) goes outside of the flying area or if any collision happens during the task.

1.2 Task Variations

Since the difficulty of the control problem depends on the level of sensor noise and wind disturbance, we provide three versions of the task with increasing level of difficulty:

- ***1A*** *noiseless:* the dynamics of the quadrotors is deterministic and the state returned is the true platform state;
- ***1B*** *noisy:* the dynamics of the quadrotors is stochastic and the state returned is a noisy estimate of the platform state (i.e. with additional white noise);
- ***1C*** *noisy and windy:* the dynamics of the quadrotors is stochastic and affected by wind disturbances (following a wind model), the state returned is a noisy estimate of the platform state (i.e. with additional correlated noise).

If there is sufficient interest additional variations could be added, these includes (but are not limited to):

- *obstacles:* e.g. cylinders in the flying area which are impassable and result in crashes. The obstacles would be represented by providing the coordinates of their center and radius, for example.
- *multitask:* a multitask setting which involves repeated attempts at tasks drawn from some distribution.

[5] We refer to the QRSim manual `http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf` for detail about the helicopter transition dynamics.

[6] While the control law that governs the quadrotor attempts to maintain a fix maximum speed, in practice the true speed will not be constant due to sensor noise wind disturbance and dynamic effects.

[7] In specific the 2D velocity action for the mouse at time t is computed as:

$$a_t^m = V_M \frac{\mathtt{v}_t}{\|\mathtt{v}_t\|} \quad \text{where} \quad \mathtt{v}_t = \sum_{i=1}^{N} \frac{\begin{bmatrix} p_x^i \\ p_y^i \end{bmatrix}_t - \begin{bmatrix} p_x^m \\ p_y^m \end{bmatrix}_t}{\left\| \begin{bmatrix} p_x^i \\ p_y^i \end{bmatrix}_t - \begin{bmatrix} p_x^m \\ p_y^m \end{bmatrix}_t \right\|^2}$$

and V_M is the (user-configurable) maximum speed that the mouse can achieve.

[8] $d_{2D}(x_T^i, x_T^m) = [p_x^i, p_y^i] \cdot [p_x^m, p_y^m]^\intercal$.

[9] The default value of the negative reward is -1000 but it can be configured by the user.

1.3 Simulation Code

All the ingredients of the scenario described above are implemented as a task class for the quadrotor simulator QRSim[10]; the three variations of the scenario are named:

- *1A*: `TaskCatsMouseNoiseless.m`,
- *1B*: `TaskCatsMouseNoisy.m`,
- *1C*: `TaskCatsMouseNoisyAndWindy.m`.

We also provide the example file `main_catsmouse.m` which shows how to initialize and run a task, how to retrieve the platform state and issue actions.

The task, the configurations and the example main files are in the directory `scenarios/catsmouse` within the QRSim simulator.

Note: Within the simulator the helicopter state x^i is denoted as eX:

$$eX = [\tilde{p}_x, \tilde{p}_y, \tilde{p}_z, \tilde{\phi}, \tilde{\theta}, \tilde{\psi}, 0, 0, 0, \tilde{p}, \tilde{q}, \tilde{r}, 0, \tilde{a}_x, \tilde{a}_y, \tilde{a}_z, h, \dot{p}_x, \dot{p}_y, \dot{h}]^\intercal$$

while the actions a^i are denoted as controls u:

$$u = [v_x, v_y]^\intercal$$

with the variables defined in section A.

[10] We refer to the QRSim manual for details on the simulator API `http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf`.

Chapter 2

Scenario 2: Search and Rescue

This scenario is designed to expose the complex interplay between sensing and acting typical in autonomous robotics task.

Several targets (people) are lost/injured on the ground in a landscape and need to be located and rescued. A helicopter agent is equipped with a camera/classification module for predicting the position of targets in its field of vision, but the quality of predictions depends upon the ground type and the geometry between helicopter and ground (e.g. the distance). Rather than raw images the camera module provides higher-level data in the form of log likelihood differences for the current observation conditioned on the presence or absence of a target.

2.1 MDP

The underlying system is modelled as a discrete time MDP on a continuous state space. The system is updated at a frequency of 1Hz[1].

States: the state $s_t = (x_t, b_t)$ at time t comprises the helicopter data x_t which includes the agents position, velocity and orientation, and b_t^i, for $i \in \{1, ..., N\}$ the position of N targets. All are continuous variables. We assume that the ground is flat and that the targets are located on the ground.

Dynamics: Markovian transition dynamics for the helicopter[2] are defined by $P(x'|x, a)$ which denotes the conditional density of state x' at $t + 1$ given $(x_t, a_t) = (x, a)$ at time t. These dynamics are independent of the targets b_t^i.

Targets b_t^i are stationary unless a helicopter hovers (i.e. its speed is low) sufficiently close to it, in which case the target is removed from the environment. More formally:

$$if\ \exists\ i\ :\ d_{3D}\left(x_t, b_t^i\right) < \delta\ \wedge\ \parallel [u, v, w]_t^\intercal \parallel < \epsilon \quad \Rightarrow \quad N = N - 1 \qquad i \in \{1, ..., N\} \tag{2.1}$$

where d_{3D} is the standard Euclidean distance[3] and (ϵ, δ) are user specified thresholds[4].

[1] The update rate is user-configurable with default value of 1Hz.

[2] We refer to the QRSim manual `http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf` for detail about the helicopter transition dynamics.

[3] Defined as

$$d_{3D}(x_t, b_t^i) = \parallel b_t^i - [p_x, p_y, p_z]_t^\intercal \parallel .$$

[4] The default values for the distance threshold and the speed threshold are $\delta = 5m$ and $\epsilon = 0.1m/s$

Actions: the action a_t at time t is expressed in terms of a 3D velocity vector in global NED coordinates:

$$a_t = [v_x, v_y, v_z]_t^\intercal. \tag{2.2}$$

Is worth remembering that the actions are nothing more than set points to a PID controller that attempts to drive that UAV at the requested velocity. Due to delays and disturbances mismatches between the commanded and the actual velocity of the UAV have to be expected.

Rewards: According to the way a target is rescued the reward r_t at time t is defined to be 1 when a helicopter hovers sufficiently close to a target and a small negative value otherwise. More formally:

$$r_t = \begin{cases} 1 & if\ \exists\ i\ :\ d_{3D}\left(x_t, b_t^i\right) < \delta\ \wedge\ \parallel [u,v,w]_t^\intercal \parallel < \epsilon \qquad i \in \{1, ..., N\} \\ -1/T & otherwise \end{cases} \tag{2.3}$$

for a task duration T and the same ϵ, δ that define a person as rescued. A large negative reward[5] is returned if the helicopter goes outside of the flying area or if any collision happens during the task.

2.2 Observations

The helicopter state x_t is observed directly although depending on the specific task variation (see Section 2.3) the state variables might be affected by stochastic noise.

The position of the targets b_t^i is not known, but observations o_t are provided by the camera at each time step.

Following standard object detection techniques we assume that the incoming image is split into M windows $\{w_t^j\}_{j=1}^M$ (of size informed by the current altitude and an assumed fixed size for the person) which are then analysed for targets. Given the current UAV pose x_t and the fixed camera parameters the set of windows re-projects to a set of M patches $\{g_t^j\}_{j=1}^M$ on the ground[6] with centres $\{c_t^j\}_{j=1}^M$. The observation o_t is simply the collection of the log likelihood ratios for each of the ground patches g_t^j:

$$o_t = \left\{ \log \left(\frac{\Pr(\text{image at time } t \mid \text{target in } g_t^j, \text{ agent at } x_t)}{\Pr(\text{image at time } t \mid \text{no target in } g_t^j, \text{ agent at } x_t)} \right) \right\}_{j=1}^{M}.$$

2.3 Task Variations

The difficulty of solving the task depends on the number of platforms that are employed as well as on the level of sensor noise and wind disturbance; we provide four versions of the task with increasing level of difficulty:

[5] The default value of the negative reward is -1000 but it can be configured by the user.

[6] Here we assume that everything needed to provide the link between the ground coordinates and the camera's frame is engineered by hand and does not have to be learned autonomously. In other words it is assumed that a map is available and known and the mapping

$$\{g_t^j\}_{j=1}^M \mapsto \{w_t^j\}_{j=1}^M$$

is known, but does not have to be considered explicitly by the agent.

- ***2A*** *single helicopter noiseless:* only one helicopter is used for the search, its dynamic is deterministic and the state returned is the true platform state;
- ***2B*** *single helicopter noisy:* only one helicopter is used for the search, its dynamics is stochastic and the state returned is a noisy estimate of the platform state (i.e. with additional correlated noise);
- ***2C*** *multiple helicopters noiseless:* several helicopters are used for the search, their dynamics is deterministic and the state returned is the true platform state;
- ***2D*** *multiple helicopters noisy and windy:* several helicopters are used for the search, their dynamics is stochastic and affected by wind disturbances (following a wind model), the state returned is a noisy estimate of the platform state (i.e. with additional correlated noise).

2.4 Simulation Code

All the ingredients of the scenario described above are implemented as task classes for the QRSim quadrotor simulator; the four variations of the scenario are named:

- *2A*: `TaskSearchRescueSingleNoiseless.m`,
- *2B*: `TaskSearchRescueSingleNoisy.m`,
- *2C*: `TaskSearchRescueMultipleNoiseless.m`
- *2D*: `TaskSearchRescueMultipleNoisyAndWindy.m`.

We also provide the example file `main_searchrescue.m` which shows how to initialize and run a task, how to retrieve the platform state, retrieve the observations and issue actions.

The task, the configurations and the example main files are in the directory `scenarios/searchrescue` within the QRSim simulator.

Note: Within the simulator the helicopter state x^i is denoted as eX:

$$eX = [\tilde{p}_x, \tilde{p}_y, \tilde{p}_z, \tilde{\phi}, \tilde{\theta}, \tilde{\psi}, 0, 0, 0, \tilde{p}, \tilde{q}, \tilde{r}, 0, \tilde{a}_x, \tilde{a}_y, \tilde{a}_z, h, \dot{p}_x, \dot{p}_y, \dot{h}]^\intercal$$

while the actions a^i are denoted as controls u:

$$u = [v_x, v_y, v_z]^\intercal$$

with the variables defined in appendix A.

Chapter 3

Scenario 3: Plume modelling

Several smoke plumes evolve over time and a helicopter agent is equipped with a sensor that measures the concentration of smoke. The plume follows a known model but with unknown parameter values. The objective is to provide a smoke concentration estimate $\hat{c}_T$ at some pre-specified time T[1].

3.1 MDP

The underlying system is modelled as a discrete time Markov process on a continuous state space. The system is updated at a frequency of 1Hz[2].

States: the state $s_t = (x_t, c_t)$ at time t comprises the helicopter data x_t which includes the agents position, velocity and orientation, which are continuous variables, and c_t the smoke concentration over the whole flight volume, also continuous.

Actions: The action a_t at time t is expressed in terms of a velocity vector in global NED coordinates:

$$a_t = [v_x, v_y, v_z]_t^\intercal. \tag{3.1}$$

Dynamics: Markovian transition dynamics for the helicopter[3] and the smoke concentration are defined by $P(s'|s, a)$ which denotes the conditional density of state s' at $t+1$ given $(s_t, a_t) = (s, a)$ at time t. The smoke evolves according to a plume model (see section 3.2 for details) and its evolution is assumed to be independent of the helicopter actions and state $P(c'|c, x, a) = P(c'|c)$.

Observations: The helicopter state x_t is observed directly although depending on the specific task variation (see Section 3.2). The smoke concentration c_t is not known, but noisy observations o_t are provided by a concentration sensor at each time step returning the concentration at the position in which the helicopter is located.

[1] For simplicity we refer to concentration of smoke, in a real environment this would be a property of the plume that can be realistically measured e.g. CO concentration.

[2] The update rate is user-configurable with default value of 1Hz.

[3] We refer to the QRSim manual `http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf` for detail about the helicopter transition dynamics.

Reward: For the tasks 3A, 3B, 3C and 3D (see section 3.2) in which the concentration is static, the agent must provide a set of concentration estimates $\{\hat{c}_T^j\}_{j=1}^M$ at a series of M spatial locations specified by the task. Given the $\{\hat{c}_T^j\}_{j=1}^M$ the performance is simply computed as (minus) the square error between the true concentrations and the estimates provided by the agent:

$$r_T = -\sum_{j=1}^{M} |c_T^j - \hat{c}_T^j|^2.$$

For tasks 3E, 3F and 3G in which the concentration evolves in time, each $\hat{c}_T^j$ can be though of as a random variable with an associated probability distribution $\Pr(\hat{c}_T^j)$. The performance is computed as the KL divergence from the true concentration distribution[4] $\Pr(c_T)$ and the provided estimate $\Pr(\hat{c}_T)$:

$$r_T = KL(\Pr(c_T^1, .., c_T^M) \| \Pr(\hat{c}_T^1, .., \hat{c}_T^M)).$$

3.2 Task Variations

The complexity of the estimation problem changes substantially depending on the type of dispersion model followed by the plume, and on the number of helicopters used to tackle the task hence we provide several version of the task with increasing level of difficulty:

- ***3A*** *single source static Gaussian concentration:* only one helicopter is used for the sampling, its dynamic is deterministic, the state returned is the true platform state, the smoke concentration is static and has the form or a three dimensional Gaussian centred at the source (see equation B.3).
- ***3B*** *single source static Gaussian dispersion model:* only one helicopter is used for the sampling, its dynamics is stochastic and affected by wind disturbances (following a wind model), the state returned is a noisy estimate of the platform state (i.e. with additional correlated noise) and the smoke concentration is static and has the form specified by what is commonly called a Gaussian dispersion model (see equation B.4).
- ***3C*** *multiple sources static Gaussian dispersion model:* only one helicopter is used for the sampling, its dynamics is stochastic and affected by wind disturbances (following a wind model), the state returned is a noisy estimate of the platform state (i.e. with additional correlated noise) and the smoke concentration is static and has the form specified by the superposition of several sources each of which follows a Gaussian dispersion model (see equation B.5).
- ***3D*** *multiple helicopters multiple sources static Gaussian dispersion model :* as above but multiple helicopters are used for the sampling.
- ***3E*** *single source time-varying Gaussian puff dispersion model:* only one helicopter is used for the sampling, its dynamics is stochastic and affected by wind disturbances (following a wind model), the state returned is a noisy estimate of the platform state (i.e. with additional correlated noise) and the smoke concentration is time-varying and has the form specified by what is commonly called a Gaussian puff dispersion model (see equation B.6).

[4] In practice in order to enable the computation of the reward the agent will be asked to repeatedly return (i.e. draw samples from its distribution) the value of $\hat{c}_t$ at the locations specified by the task.

- ***3F*** *multiple sources time-varying Gaussian puff dispersion model:* only one helicopter is used for the sampling, its dynamics is stochastic and affected by wind disturbances (following a wind model), the state returned is a noisy estimate of the platform state (i.e. with additional correlated noise) and the smoke concentration is static and has the form specified by the superposition of several sources each of which follows a Gaussian puff dispersion model (see equation B.7).

- ***3G*** *multiple helicopters multiple sources time-varying Gaussian puff dispersion model :* as above but multiple helicopters are used for the sampling.

Note: Since the dispersion models are known, one possible way to solve the tasks is to estimate the model parameters (or a distributions over them); while this is an appropriate solution we emphasize that the agent is not required to solve the task in this way and so other forms for the concentration (or for the distribution over the concentration) are equally appropriate.

3.3 Simulation Code

All the ingredients of the scenario described above are implemented[5] as a task class for the quadrotor simulator QRSim[6]; the seven variations of the scenario are named:

- *3A*: `TaskPlumeSingleSourceGaussian.m`,
- *3B*: `TaskPlumeSingleSourceGaussianDispersion.m`,
- *3C*: `TaskPlumeMultiSourceGaussianDispersion.m`,
- *3D*: `TaskPlumeMultiHeliMultiSourceGaussianDispersion.m`,
- *3E*: `TaskPlumeSingleSourceGaussianPuffDispersion.m`,
- *3F*: `TaskPlumeMultiSourceGaussianPuffDispersion.m`,
- *3G*: `TaskPlumeMultiHeliMultiSourcePuffDispersion.m`.

We also provide the example file `main_plume.m` which shows how to initialize and run a task, how to retrieve the platform state, retrieve the observations, retrieve the location at which to provide estimates, issue actions and return concentration estimates.

The task, the configurations and the example main files are in the directory `scenarios/plume` within the QRSim simulator.

Note: Within the simulator the helicopter state x^i is denoted as eX:

$$eX = [\tilde{p}_x, \tilde{p}_y, \tilde{p}_z, \tilde{\phi}, \tilde{\theta}, \tilde{\psi}, 0, 0, 0, \tilde{p}, \tilde{q}, \tilde{r}, 0, \tilde{a}_x, \tilde{a}_y, \tilde{a}_z, h, \dot{p}_x, \dot{p}_y, \dot{h}]^\intercal$$

while the actions a^i are denoted as controls u:

$$u = [v_x, v_y, v_z]^\intercal$$

with the variables defined in section A.

[5] We are currently in the process of finalizing the implementation.

[6] We refer to the QRSim manual for details on the simulator API `http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf`.

Appendix A

Variable Definitions

Helicopter state variables common to all the tasks:

p_x	true x position (NED coordinates)	m
p_y	true y position (NED coordinates)	m
$\tilde{p}_x$	x position estimate from GPS (NED coordinates)	m
$\tilde{p}_y$	y position estimate from GPS (NED coordinates)	m
$\tilde{p}_z$	z position estimate from GPS (NED coordinates)	m
$\tilde{\phi}$	roll attitude in Euler angles right-hand ZYX convention	rad
$\tilde{\theta}$	pitch attitude in Euler angles right-hand ZYX convention	rad
$\tilde{\psi}$	yaw attitude in Euler angles right-hand ZYX convention	rad
$\tilde{p}$	rotational velocity around x body axis from gyro	rad/s
$\tilde{q}$	rotational velocity around y body axis from gyro	rad/s
$\tilde{r}$	rotational velocity around z body axis from gyro	rad/s
$\tilde{a}_x$	linear acceleration in x body axis from accelerometer	m/s^2
$\tilde{a}_y$	linear acceleration in y body axis from accelerometer	m/s^2
$\tilde{a}_z$	linear acceleration in z body axis from accelerometer	m/s^2
h	altitude[1]from altimeter NED	m
$\dot{p}_x$	x velocity from GPS (NED coordinates)	m/s
$\dot{p}_y$	y velocity from GPS (NED coordinates)	m/s
$\dot{h}$	altitude rate from altimeter NED	m/s
v_x	desired x velocity control (NED coordinates)	m/s
v_y	desired y velocity control (NED coordinates)	m/s

We remind the reader that NED stands for Noth-East-Down as explained in more detail in the QRSim manual.

Appendix B

Concentration Models

To interpret the following models, is useful to introduce some nomenclature:

x, y, z	coordinates w.r.t. the global NED frame of reference	m
x', y', z'	coordinates w.r.t. the wind frame of reference	m
X_s, Y_s	coordinates of the source w.r.t. the global NED frame	m
$x\prime_s, y'_s$	coordinates of the source w.r.t. the wind frame of reference	m
Q_s	emission rate of source s	Kg/s
H_s	equivalent height of source s	m
u	constant magnitude of the wind speed	m/s
S	number of sources	
a	diffusion parameter	m^{2-b}
b	diffusion parameter	
α_w	wind direction (clockwise from north)	$rads$
I_s	total number of puff for source s	
T_s^i	time at which puff i of source s was emitted	s
Q_s^i	total amount of smoke emitted by source s at time T^i	Kg
$\mathbf{\Sigma}$	Gaussian concentration covariance matrix	

We also introduce a change of reference frame, namely from global frame to wind frame (a frame of reference with origin in the global frame and aligned with the wind direction), since some of the models are expressed in this coordinates:

$$x' = x\cos(\alpha_w) \tag{B.1}$$
$$y' = y\sin(\alpha_w). \tag{B.2}$$

B.1 Single Source Gaussian Concentration Model

$$c(x,y,z) = \exp\left(-\frac{1}{2}\begin{bmatrix} x-X_s \\ y-Y_s \\ z-H_s \end{bmatrix}^T \mathbf{\Sigma}^{-1} \begin{bmatrix} x-X_s \\ y-Y_s \\ z-H_s \end{bmatrix}\right) \tag{B.3}$$

B.2 Single Source Gaussian Dispersion Model

A standard static plume dispersion (for more details see [**?**]):

$$c(x', y', z) = \frac{Q}{2\pi u a (x' - X'_s)^b} \exp\left(-\frac{(y' - Y'_s)^2}{2a(x' - X'_s)^b}\right) \left[\exp\left(-\frac{(z - H_s)^2}{2a(x' - X'_s)^b}\right) + \exp\left(-\frac{(z + H_s)^2}{2a(x' - X'_s)^b}\right)\right]. \tag{B.4}$$

B.3 Multiple Sources Gaussian Dispersion Model

In the case of multiple sources the total concentration can be computed by superposition:

$$c(x', y', z) = \sum_{s=1}^{S} c(x', y', z; X'_s, Y'_s, H_s, Q_s). \tag{B.5}$$

B.4 Single Source Gaussian Puff Dispersion Model

A standard time varying plume dispersion (for more details see [**?**]):

$$c(x', y', z, t) = \sum_{i=1}^{I} \left\{ \frac{Q_s^i}{8(\pi a (x' - X'_s)^b)^{3/2}} \exp\left(-\frac{(x' - X'_s - u(t - T_s^i))^2 + (y' - Y'_s)^2}{2a(x' - X'_s)^b}\right) \left[\exp\left(-\frac{(z - H_s)^2}{2a(x' - X'_s)^b}\right) + \exp\left(-\frac{(z + H_s)^2}{2a(x' - X'_s)^b}\right)\right]\right\}. \tag{B.6}$$

B.5 Multiple Sources Gaussian Puff Dispersion Model

Even in the case a time varying dispersion model, for multiple sources the total concentration can be computed by superposition:

$$c(x', y', z, t) = \sum_{s=1}^{S} c(x', y', z, t; X'_s, Y'_s, H_s, Q_s^{1..I_s}). \tag{B.7}$$