CompLACS Helicopters Scenarios

PRELIMINARY VERSION

February 1, 2013

Chapter 1

Quadrotors Scenarios

Abstract

This document describes the three types of scenarios devised at UCL to develop and test learning and control algorithm with application to a flock of autonomous quadrotor helicopters, one of the three real world platforms of the CompLACS project. The three application scenarios are explicitly chosen to expose different types of challenges that occur in the domain of multi-platform aerial robotics so to provide a variety of research opportunities.

This report is divided into three parts one for each of the three scenarios; each part gives a formal description of the scenario in terms of its setup, its objective, and its variations.

In order to aid the development and testing of learning algorithms, we provide a simulation environment based on the QRSim quadrotor simulator for each the scenarios along with handy code examples. Technical details specific to the scenarios implementation are reported in the appendix.

Since the scenarios simulations are built on top of our QRSim, we refer to its manual (`http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf`) for details about the simulator and its API.

Contents

1.1 Scenario 1: Cats and Mouse game

The first of the scenarios is designed to focus primarily on the challenges encountered in the coordinated control of multiple UAVs; the associated problems of sensing and state estimation are somewhat simplified by the choice of task, environment and platform sensors.

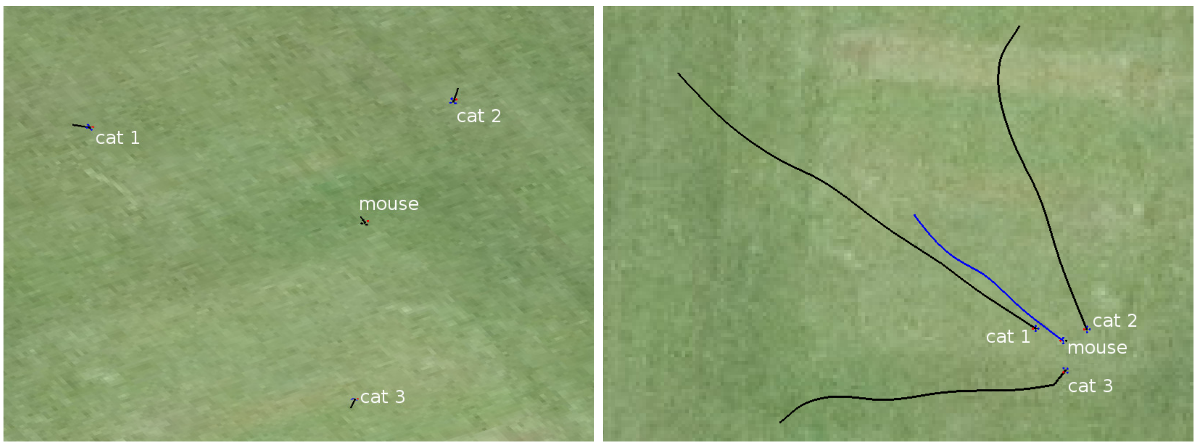
1.1.1 Description

The task to be accomplished in this scenario is in the form of a team game in which N helicopters (cats) have to surround and effectively trap (i.e. get close to) another helicopter (mouse) at the end of the allotted time.

For simplicity the task is assumed to take place in an area devoided of obstacles so that helicopters can fly freely. Getting in contact with the ground or another UAV will however produce a collision. The platforms are equipped with noisy sensors so only observation of the vehicle state are available. Depending on the circumstances in the flying area there might be present wind and other aerodynamic disturbances that affect the flight behaviour of the platforms.

Snapshots of the initial ($t = 0$) and terminal($t = T$) configurations from a typical successful run are visible in figures 1.1a and 1.1b respectively.

In the next section we give a more formal description of the problem.



(a) $t = 0$ (b) $t = T$

Figure 1.1: Typical successful run: start(a) and end(b) configurations.

1.1.2 MDP

The underlying system is modelled as a discrete time, finite horizon MDP on a continuous state space. The system runs from $t = 0$ to $t = T$ with a time step equivalent to $1s$ of simulated time[1].

State: The state $s_t = (x_t^1, ..., x_t^N, x_t^m)$ at time t comprises of the state vectors of all the cats (superscripts $1, ..., N$) and of the mouse (superscript m) helicopters. Each platform state contains in turn the position ($[p_x, p_y, p_z]^\intercal$), velocity ($[u, v, w]^\intercal$), orientation ($[\phi, \theta, \psi]^\intercal$) and rotational velocity ($[p, q, r]^\intercal$) of the platform. All of these are continuous variables (see section 1.1.4 for more details).

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

The environment is itself three dimensional although we will see (section 1.1.2) that 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.

The initial position of cats are generated as:

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

where $\alpha_i \in \mathcal{U}(0, 2\pi)$, $d_i \in \mathcal{U}(D_m, D_M)$ and $\mathcal{U}(a, b)$ indicates a uniform distribution over the interval $[a, b]$. 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 any two cats; this prevent the occurrence of an initial state in which two cats are too close.

At $t = 0$ all the helicopters are stationary (i.e. their velocities are zero).

Actions: A real quadrotor of the type considered in our task generally presents four continuous control inputs (i.e. pitch, roll, yaw angles and throttle) which needs to be updated at a rate of $50Hz$, constituting a rather large control space.

To limit the space of control inputs, in our setup each UAV is equipped with a close loop PID controller that accepts 2D linear velocity commands $a_t^i = [v_x^i, v_y^i, v_z^i]^\intercal$ (in global coordinates) and combines them with the estimated platform velocity to produce the necessary attitude and throttle commands for the platform. The PID accepts commanded velocity at a rate of $1Hz$ and provides the platform controls at $50Hz$. To additionally reduce the control space the PID takes also care of maintaining a constant altitude and heading[3].

The action space $a_t = (a_t^1, ..., a_t^N)$ at time t comprises of the 2D linear velocity commands for each of the cats helicopters.

Dynamics: Markovian transition dynamics are defined by a distribution $P(s_{t+1}|s_t, a_t)$ which denotes the conditional density of state s at time $t+1$ given state-action $(s_t, a_t) = (s, a)$ at time t.

In the case of the cats helicopters the transition dynamics is defined by the combination of PID velocity controllers, platforms dynamics, sensor and wind dynamics (since these in turn effect the quadrotor) all of which are part of the QRSim simulator[4].

For the mouse helicopter the transition dynamics is not only defined by the platform dynamics but also by the way the mouse moves in response to the cats. As the task starts the mouse helicopter tries to escape the cats by moving at a constant (max) speed[5] and choosing a direction that prioritize escaping from the closer cats. In specific the 2D velocity action for the

[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 use of a PID controller in addition to reducing the number of control inputs makes the task closer to what is possible to implement and test using real quadrotors.

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

[5]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.

mouse at time t is computed as:

$$a_t^m = V_M \frac{\mathsf{v}_t}{\|\mathsf{v}_t\|} \quad \text{where} \quad \mathsf{v}_t = \sum_{i=1}^N \frac{\begin{bmatrix} p_x^i \\ p_y^i \end{bmatrix}_t - \begin{bmatrix} p_x^m \\ p_y^m \end{bmatrix}_t}{\left\| \begin{bmatrix} p_x^i \\ p_y^i \end{bmatrix}_t - \begin{bmatrix} p_x^m \\ p_y^m \end{bmatrix}_t \right\|^2} \tag{1.1}$$

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

Different control strategies could be considered for the mouse; in practise the one considered in equation 1.1 is both simple and has shown to be sufficient to provide for a challenging task.

Observations The helicopter state x_t^i is observed directly although depending on the specific task variation (see Section 1.1.3) the state variables might be affected by stochastic noise. For more details about the noise models used by QRSim we refer the reader to the simulator manual (`http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf`).

Reward: The cats are successful if the are able to trap the mouse at the end of the task; a meaningful final reward can be defined as the sum of the squared (2D) distances[6] between the cats and the mouse at time T:

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

A large negative reward[7] is returned if any of the helicopters (including the mouse) goes outside of the flying area or if any collision happens during the task.

1.1.3 Task Variations

The difficulty of the considered control problem depends on the level of sensor noise and wind disturbance; so we define three versions of the task with increasing level of realism (and consequently 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 correlated 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).

1.1.4 Simulation Code

The cat and mouse scenario can be promptly defined on top of the QRSim simulator by means of dedicated task classes[8]. In specific we make available one task class for each of the three scenario variations introduced in section 1.1.3:

[6] $d_{2D}(x_T^i, x_T^m) = \|[p_x^i, p_y^i]_T^\intercal - [p_x^m, p_y^m]_T^\intercal\|$.

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

[8] We refer to the `http://complacs.cs.ucl.ac.uk/complacs/simulator/manual.pdf` for details on task classes.

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

Commenting the code in details is beyond the scope of this report but it is useful to briefly outline which task methods are responsible for handling the various task specific components:

- `init()`: defines all the platforms and sensor parameters;
- `reset()`: defines the UAVs starting condition;
- `step(U)`: defines the mouse evasion strategy and uses the PID controllers to transform velocity commands into angle command to the helicopters;
- `reward()`: defines the task reward.

Listing 1.1 (file `main_catsmouse.m`) shows the set of steps that are necessary to run a scenario task.

After the desired task is initialized (line 4), a basic for loop is executed for the number of timesteps specified by the task. Within the loop the 2D velocity control is computed for each helicopter and passed to the corresponding PID (line 37). In our listing we show a simple (and suboptimal) scheme in which the velocity direction of each cat directly towards the future mouse position mouse (line 20-23) but that also keeps away from neighbouring cats (line 26-34). The helicopter control input produced by the PID controllers are then used to step the simulator (line 41). Finally after the execution of the task is concluded, the final reward for the task can be retrieved (line 46).

We remind the reader that within the simulator the helicopter state x^i is denoted as `eX`:

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

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

$$\texttt{u} = [v_x, v_y]^\intercal$$

with the variables defined in section 1.4.

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

Listing 1.1: main_catsmouse.m

```matlab
qrsim = QRSim();

% load task parameters and do housekeeping
state = qrsim.init('TaskCatsMouseNoisyAndWindy');

U = zeros(2,state.task.Nc);

% run the scenario and at every timestep generate a control
% input for each of the uavs
for i=1:state.task.durationInSteps,

    % get the mouse position (note id state.task.Nc+1)
    mousePos = state.platforms{state.task.Nc+1}.getEX(1:2);

    % quick and easy way of computing velocity controls for each cat
    for j=1:state.task.Nc,
        collisionDistance = state.platforms{j}.getCollisionDistance();

        % vector to the mouse
        u = mousePos - state.platforms{j}.getEX(1:2);

        % add a weighted velocity (i.e. "predict" where the mouse will be)
        u = u  + (norm(u)/2)*state.platforms{state.task.Nc+1}.getEX(18:19);

        % keep away from other cats if closer than 2*collisionDistance
        for k = 1:state.task.Nc,
            if(state.platforms{k}.isValid())
                d = state.platforms{j}.getEX(1:2)
                    - state.platforms{k}.getEX(1:2);
                if((k~=j)&&(norm(d)<2*collisionDistance))
                    u = u + (1/(norm(d)-collisionDistance()))*(d/norm(d));
                end
            end
        end

        % scale by the max allowed velocity
        U(:,j) = state.task.velPIDs{j}.maxv*(u/norm(u));
    end

    % step simulator
    qrsim.step(U);
end

% get final reward
fprintf('final reward: %f\n',qrsim.reward());
```

1.2 Scenario 2: Search and Rescue

The second scenario is designed explicitly to expose the complex interplay between sensing and acting typical in autonomous robotics task. To solve the task the agent/s has to perform inference about the state of the environment given some observations and take actions based on its belief.

1.2.1 Description

The task to be accomplished in this scenario is in the form of a wilderness search and rescue mission; several targets (people) are lost/injured on the ground in a landscape and need to be located and rescued.

In addition to its navigation sensors each helicopter agent taking part in the search is equipped with a camera/classification module that allows it to detect the presence of targets in its field of view. 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. The quality of detection depends upon the ground type and the geometry between helicopter and ground (e.g. the distance).

The search era is limited but its extent is so that a trivial lawn mower pattern of search will not allow to cover all the area in the allotted time. The landscape has different types of terrain; persons are more likely to be present on some class of terrain than others. For simplicity the persons are assumed to not move during the task.

Additionally we assume that the flying area is free of obstacles so that the UAVs can move freely in the 3D space. Getting in contact with the ground or another UAV will however produce a collision.

A snapshot from a typical run is shown in figure 1.2 along with an example of the observation returned by the camera/classifier model.

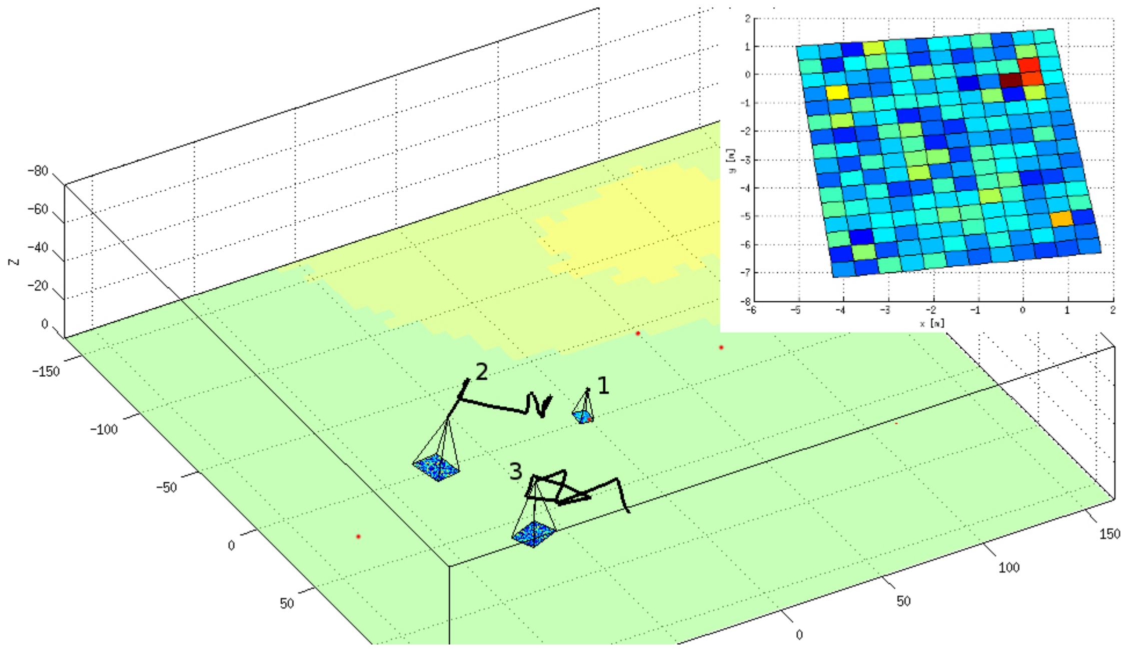


Figure 1.2: Search and rescue run with three helicopters; note the different terrain types (denoted by different colors) and the persons (red dots). The insert shows the camera observation from UAV 1 that happens to be over a person.

1.2.2 MDP

The underlying system is modelled as a discrete time, finite horizon MDP on a continuous state space. The system runs from $t = 0$ to $t = T$ with a time step equivalent to $1s$ of simulated time[9].

States: The state $s_t = (x_t^1, ..., x_t^N, b_t^1, ..., b_t^P)$ at time t comprises the helicopters state vectors x_t^i and the location of the P targets b_t^j.

Each platform state contains the position ($[p_x, p_y, p_z]^\intercal$), velocity ($[u, v, w]^\intercal$), orientation ($[\phi, \theta, \psi]^\intercal$) and rotational velocity ($[p, q, r]^\intercal$) of the platform. The environment is itself three dimensional and the helicopter can freely move in three dimensions.

The person's location is expressed in terms of its coordinate w.r.t. the global reference frame $b^j = [p_x, p_y, 0]^\intercal$; the persons' z coordinate is zero since we assume that the ground is flat and that the targets are located on the ground.

Initial State: At the beginning of the task a new terrain map is generated based on the number of terrain classes and split between class types specified by the user[10]. The extent of the flying area is scaled accordingly to the number of platforms and time horizon in order to ensure that the task is non trivial. Subsequently a number of persons is randomly placed in the search area. The user can control where persons are placed by specifying the probability of a target to be located on a specific terrain class[11]. Finally the helicopters are located randomly (with uniform probability) around the flying area at the nominal flight height[12].

Actions: At each time step the agent specifies an action a_t for each of the UAV taking part in the task; the action is expressed in terms of a 3D velocity vector in global NED coordinates:

$$a_t = [v_x, v_y, v_z]_t^\intercal. \tag{1.2}$$

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.

Dynamics: For the UAVs the transition dynamics is defined by the combination of PID velocity controllers, platforms dynamics, sensor and wind dynamics (since these in turn effect the quadrotor) all of which are specified by the QRSim simulator. Such transition dynamics are Markovian and defined by $P(x_{t+1}|x_t, a_t)$ which denotes the conditional density of state x at $t+1$ given $(x_t, a_t) = (x, a)$ at time t. The dynamics of the UAVs are independent of the targets b_t^j.

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

$$\begin{aligned} &if\ \exists\ i,j\ :\ d_{3D}\left(x_t^i, b_t^j\right) < \delta\ \wedge\ \parallel [u^i, v^i, w^i]_t^\intercal \parallel < \epsilon \\ &\Rightarrow \quad P = P - 1 \qquad\qquad\qquad\qquad\qquad i \in \{1, ..., N\}, j \in \{1, ..., P\} \end{aligned} \tag{1.3}$$

[9]The update rate is user-configurable with default value of 1Hz.
[10]The user specifies what percentage of the total area should belong to each class.
[11]Each persons location is generated independently.
[12]The initial altitude can be configured by the user but is set to a $25m$ default value.

where d_{3D} is the standard Euclidean distance[13] and (ϵ, δ) are user specified thresholds[14].

Observations The helicopter state x_t is observed directly although depending on the specific task variation (see Section 1.2.3) the state variables might be affected by stochastic noise.

The position of the targets b_t^j 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 $\{w_t^k\}_{k=1}^M$ (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 projects on the ground to a set of M patches $\{g_t^k\}_{k=1}^M$ on the ground with centres $\{c_t^k\}_{k=1}^M$. More precisely we this assumes that a map is available and that the mapping

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

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

We assume that some form of person detection algorithm is run on each of the image windows w_t^k. As a result it is possible to evaluate the probability that such image patch was originated by a person (at the corresponding ground location g_t^k) versus the probability that w_t^k was originated by clear ground at location g_t^k; what is commonly called the likelihood ratio.

To generate likelihood ratios we use a model learned from the scores obtained by running an off the shelf person classifier on aerial images dataset collected with the quadrotor UAV in use at UCL. We refer to section 1.5 for more details about such model.

The observation o_t is simply the collection of the log likelihood ratios obtained for each of the image windows w_t^k (and therefore also for the corresponding ground patches g_t^k):

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

An example of the observation returned by our simulated scenario is visible in the insert of figure 1.2. A higher value of the ratio (red color) is noticeable for the ground patches that are at the person location; also note how the patches are conveniently expressed in ground coordinates.

Reward: The reward r_t for the task is designed to prize the UAVs for quickly rescuing targets. In this spirit 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:

$$r_t = \begin{cases} 1 & if\ \exists\, i,j\ :\ d_{3D}\left(x_t^i, b_t^j\right) < \delta\ \wedge\ \parallel [u,v,w]_t^\intercal \parallel < \epsilon \quad i \in \{1,...,N\}, j \in \{1,...,P\} \\ -1/T & otherwise \end{cases} \tag{1.4}$$

where T is the task duration T and ϵ, δ are respectively the distance and velocity thresholds that that define a person as rescued (see section 1.2.2). A large negative reward[15] is returned if the helicopter goes outside of the flying area or if any collision happens during the task.

[13]Defined as

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

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

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

1.2.3 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:

- ***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).

1.2.4 Simulation Code

A simulated version of each of the variations of this scenario are made available in the form of a task class that can be readily used within the QRSim helicopter simulator. More specifically the four task variations presented in section 1.2.3 are named:

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

Commenting the code in details is beyond the scope of this report but it is useful to briefly outline which task methods are responsible for handling the various task specific components:

- `init()`: defines all the platforms, sensor and environment parameters;
- `reset()`: defines the task starting condition for UAVs and persons;
- `step(U)`: uses the PID controllers to transform velocity commands into angle command to the helicopters, checks if a target was located and removes it;
- `reward()`: returns the reward at the current timestep;
- `getNumberOfPersons()`: returns the number of persons present at the beginning of the task.

We also provide the example file `main_searchrescue.m` (listing 1.2) which briefly shows how to initialize and run any of the search and rescue tasks. The layout of the code is very similar to what we have seen for the cats and mouse scenarios; after a preliminary initialization (lines $2-9$), a for loop takes care of stepping the simulation (line 38) for the predefined number of time steps. For simplicity in this example, the UAVs move in the space at a fix velocity changing their

Listing 1.2: main_searchrescue.m

```matlab
% create simulator object
qrsim = QRSim();

% load task parameters and do housekeeping
state = qrsim.init('TaskSearchRescueSingleNoiseless');

% create a 3 x helicopters matrix of control inputs
% column i will contain the 3D NED velocity [vx;vy;vz] for helicopter i
U = zeros(3,state.task.numUAVs);

% number of persons to locate in this task
np = state.task.getNumberOfPersons();

% run the scenario and at every timestep generate a control
% input for each of the uavs
for i=1:state.task.durationInSteps,
    % random search policy in which the helicopter(s) moves around
    % at a fixed velocity changing direction every once in a while
    if(rem(i-1,10)==0)
        for j=1:state.task.numUAVs,
            if(state.platforms{j}.isValid())
                % random velocity direction
                u(:,j) = rand(2,1)-[0.5;0.5];
                % fixed velocity 0.5 times max allowed velocity
                U(:,j) = [0.5*state.task.velPIDs{j}.maxv*...
                          (u(:,j)/norm(u(:,j)));0];

                % if the uav is going astray we point it back to the center
                p = state.platforms{j}.getEX(1:2);
                if(norm(p)>100)
                    U(:,j) = [-0.8*state.task.velPIDs{j}.maxv*p/norm(p);0];
                end
            end
        end
    end

    % step simulator
    qrsim.step(U);

    % get camera measurements:
    % the output is an object of type CemeraObservation, i.e.
    % a simple structure containing the fields:
    % llkd       log-likelihood difference for each gound patch
    % wg         list of corner points for the ground patches
    % gridDims   dimensions of the grid of measurements    %
    for j=1:state.task.numUAVs,
       m = state.platforms{j}.getCameraOutput();
    end

    % reward (1 as soon as we hovering close enough to a person)
    r = qrsim.reward();
end
```

direction randomly every 10 time steps (lines $19 - 35$). For each UAV the camera measurement can be retrieved after stepping the simulator with a new action (line 47). In a more interesting policy than the random one we are showing in this example an agent would make use of this new observation (and of the previous) to decide its next action. Lastly line 51 shows how to retrieve the reward at the current time step.

In some situations it might be useful to know the total number of targets present in the environment, line 12 shows how is possible to do that using the `getNumberOfPersons()` method.

We remind the reader that within the simulator the helicopter state x^i is denoted as `eX`:

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

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

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

with the variables defined in section 1.4.

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

1.3 Scenario 3: Plume Modelling

In the third and last scenario we design tasks in which sensing and estimation of the target environment play a crucial role. In specific the actions of the agent are aimed at generating reliable predictions about the flying area.

1.3.1 Description

In this scenario we envisage the situation in which a plume is dispersed in the flying area and we are interested in using one or more UAVs to estimate the plume concentration[16].

Depending on the task settings (see section 1.3.3) one or more sources might be present in the environment and the plume distribution might evolve over time. Any wind affects both the the UAV flight behaviour and the plume distribution. UAVs are equipped with noisy navigation sensors that allow to observe their location, attitude and velocities and also with a sensor that produces noisy observation of the plume concentration.

The plume concentration follows a known model but with unknown parameter values; the task objective is to provide a concentration estimate $\hat{c}_T$ at some pre-specified time T. For this scenario we selected three main classes of concentration models; they were chosen for being widely adopted ([**?**]) while remaining relatively simple:

- *Gaussian*: when no wind is present in the flying area and the source emits plume with a constant rate, the plume disperses around the source and its concentration can be described by a three dimensional Gaussian (equation 1.8). Since the source emits at constant rate the resulting dispersion pattern is static, however to make the model not completely trivial we assume that the dispersion is not necessarily isotropic; Figure1.3a depicts an example of type of concentration pattern produced by this model.

- *Gaussian Dispersion*: when wind is present in the flying area, and the source emits plume with a constant rate, the plume is blown downwind and disperses as it gets farther from the source. The concentration takes a cone like shape which expands as the distance from the source increases (equation 1.9). In the case in which the mean wind velocity and direction do not change with time[17], the resulting concentration is also time invariant. An example of the resulting dispersion is visible in Figure1.3b.

- *Gaussian Puff Dispersion* when wind is present in the flying area but the source emits plume for short time intervals (i.e short puffs), each puff travels downwind expanding as it gets farther from the source. At every time instant the resulting concentration is the superposition of the many puffs (equation 1.11); the concentration is therefore time variant. Figure 1.3c shows an example of such dispersion model.

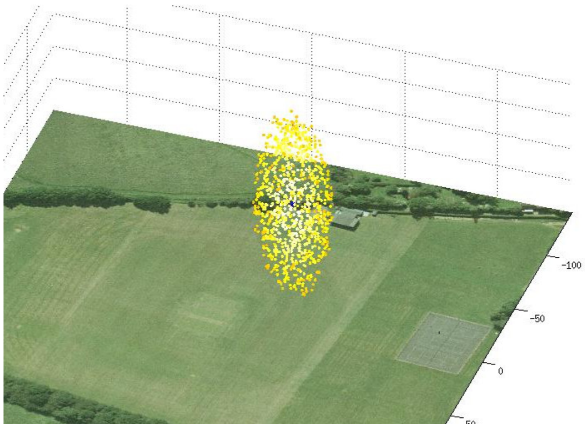
For each type of dispersion model the case in which more than one source is present in the environment can be considered simply by superimposing the effects of several individual sources; superposition gives origin to substantially more complex distributions.

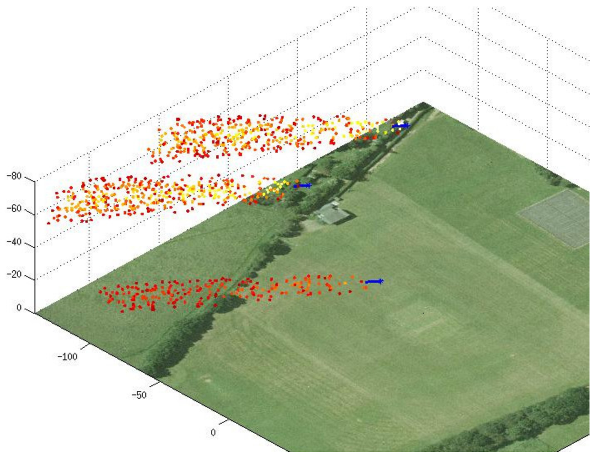
As in the previous tasks we assume that the flying area is free of obstacles so that the UAVs can move freely in the 3D space. Getting in contact with the ground or another UAV will however produce a collision.

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

[16]Hereby we simply refer to concentration, in a real environment this would be a property of the plume that can be realistically measured e.g. CO concentration.

[17]This will be the case in our settings.


(a) Single source Gaussian concentration.


(b) Multiple sources Gaussian dispersion.

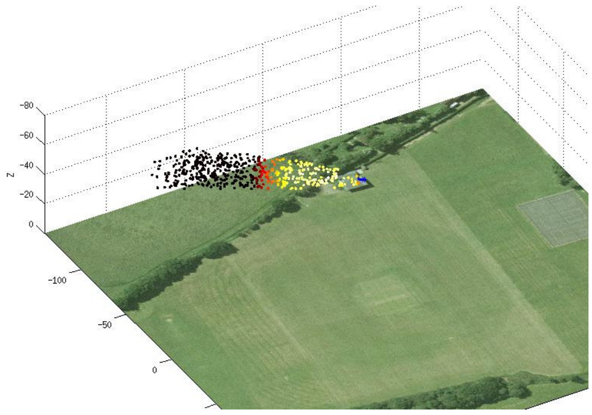

(c) Single source Gaussian puff dispersion.

Figure 1.3: Plume models; darker markers indicate lower concentration values.

solution we emphasize that the agent is not required to solve the task in this way and so model agnostic way of producing concentration estimations are equally appropriate.

1.3.2 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[18] and the task has a duration T.

States: the state $s_t = (x_t^1, .., x_t^N, c_t)$ at time t comprises the helicopter/s state x_t^i and the plume concentration c_t. Each platform state contains the position ($[p_x, p_y, p_z]^\intercal$), velocity ($[u, v, w]^\intercal$), orientation ($[\phi, \theta, \psi]^\intercal$) and rotational velocity ($[p, q, r]^\intercal$) of the platform. The environment is itself three dimensional and the helicopter can freely move in three dimensions. The plume concentration is defined at every 3D location over the whole flight volume.

Initial State: At the beginning of the task a new set of source locations within the flying area is defined with their associated emission rate so that given the current wind, the complete plume distribution can be defined. The user can control the number, height and emission rate of

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

sources through task parameters. At $t = 0$ the helicopters are located randomly (with uniform probability) in the fly area at the nominal flight height[19].

Actions: At each time step the agent specifies an action a_t for each of the UAV taking part in the task; the action is expressed in terms of a 3D velocity vector in global NED coordinates:

$$a_t = [v_x, v_y, v_z]_t^\intercal. \tag{1.5}$$

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.

Dynamics: For the UAVs the transition dynamics is defined by the combination of PID velocity controllers, platforms dynamics, sensor and wind dynamics (since these in turn effect the quadrotor) all of which are specified by the QRSim simulator. Such transition dynamics are Markovian and defined by $P(x_{t+1}|x_t, a_t)$ which denotes the conditional density of state x at $t+1$ given $(x_t, a_t) = (x, a)$ at time t.

In the case of non puff-like dispersion models the plume distribution is stationary and therefore does not evolve during the task. For plume distributions defined by the Gaussian puff dispersion model puffs are emitted at random intervals drawn from an exponential distribution. Each puff travels downwind at the mean wind speed and simultaneously expands as specified by equations 1.11-1.12.

For simplicity any effect that a UAV might have on the plume concentration is neglected and the evolution in time of the plume is assumed to be independent of the helicopter actions and state $P(c_{t+1}|c_t, x_t, a_t) = P(c_{t+1}|c_t)$.

Observations: The helicopter state x_t is observed directly although depending on the specific task variation (see Section 1.3.3) the state variables might be affected by stochastic noise.

The plume concentration c_t is not known; at each time step a noisy observations o_t at the position in which the helicopter is located is returned by a concentration sensor. At present the sensor reading are corrupted by additive Gaussian noise.

Reward: The task objective is to provide a concentration estimate $\hat{c}_T$ at some pre-specified time T (i.e. at the end of the task).

For the tasks in which the concentration is stationary (i.e. tasks 3A, 3B, 3C and 3D), the agent must provide a set of concentration estimates $\{\hat{c}_T^j\}_{j=1}^M$ at a series of M spatial locations specified at the beginning of the task. Given the $\{\hat{c}_T^j\}_{j=1}^M$ the task reward 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 in which the concentration evolves over time (i.e. tasks 3E, 3F and 3G), the concentration at each location $\hat{c}_T^j$ can be though of as a random variable with an associated probability distribution $\Pr(\hat{c}_T^j)$. The better the agent is at approximating such distribution, the higher should be its reward. A simple way to compute the reward is in the form of (minus) the

[19]The initial altitude can be configured by the user but is set to a $25m$ default value.

KL divergence between true concentration distribution $\Pr(c_T^1, .., c_T^M)$ and the estimate provided by the agent[20] $\Pr(\hat{c}_T^1, .., \hat{c}_T^M)$:

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

1.3.3 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:

- ***3A** single source static Gaussian concentration:* only one helicopter is used for the sampling, its dynamic is deterministic, the navigation sensors return the true platform state, the plume concentration is static and has the form or a three dimensional Gaussian centred at the source (see equation 1.8).

- ***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 navigation sensors return a noisy estimate of the platform state (i.e. with additional correlated noise) and the plume concentration is static and has the form specified by what is commonly called a Gaussian dispersion model (see equation 1.9).

- ***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 navigation sensors return a noisy estimate of the platform state (i.e. with additional correlated noise) and the plume concentration is static and has the form specified by the superposition of several sources each of which follows a Gaussian dispersion model (see equation 1.10).

- ***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 navigation sensors return a noisy estimate of the platform state (i.e. with additional correlated noise) and the plume concentration is time-varying and has the form specified by what is commonly called a Gaussian puff dispersion model (see equation 1.11).

- ***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 navigation sensors return a noisy estimate of the platform state (i.e. with additional correlated noise) and the plume 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 1.12).

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

[20]In practice in order to enable the empirical computation of the reward the agent will be asked to return several samples of the concentration at each of the M locations specified by the task.

1.3.4 Simulation Code

All the ingredients of the scenario described above are implemented[21] as a task class for the quadrotor simulator QRSim[22]; 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.

We remind the reader that 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 1.4.

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

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

Listing 1.3: main_plume.m

```
% create simulator object and load task parameters
qrsim = QRSim();
state = qrsim.init('TaskPlumeSingleSourceGaussianDispersion');

% create a 3 x helicopters matrix of NED velocity inputs
U = zeros(3,state.task.numUAVs);

% allocate up a matrix to store all the concentration measurements
plumeMeas = zeros(state.task.numUAVs, state.task.durationInSteps);

% get the list of location the agent needs to return predictions at.
positions = state.task.getLocations();

% number of predictions the agent needs to return for each location,
% will be > 1 only if the concentration is time variant i.e. a puff model
samplesPerLocation = state.task.getSamplesPerLocation();

% run the scenario and at every timestep generate a control for each uav
for i=1:state.task.durationInSteps,
    % random exploration policy in which the helicopter(s) moves around
    % at a fixed velocity changing direction every once in a while
    if(rem(i-1,10)==0)
        for j=1:state.task.numUAVs,
            if(state.platforms{j}.isValid())
                % random velocity direction
                u(:,j) = rand(2,1)-[0.5;0.5];
                % scale by the max allowed velocity
                U(:,j) = [0.5*state.task.velPIDs{j}.maxv...
                         *(u(:,j)/norm(u(:,j)));0];
                % if the uav is going astray we point it back to the center
                p = state.platforms{j}.getEX(1:2);
                if(norm(p)>100)
                    U(:,j) = [-0.8*state.task.velPIDs{j}.maxv*p/norm(p);0];
                end
            end
        end
    end
    % step simulator
    qrsim.step(U);

    % get plume measurement
    for j=1:state.task.numUAVs,
        plumeMeas(j,i)=state.platforms{j}.getPlumeSensorOutput();
    end
end
% matrix containing all the predictions made by the agent
% for the purpose of illustration we generate those randomly
samples = randn(samplesPerLocation,size(positions,2));

% pass the sample to the task so that a reward can be computed
state.task.setSamples(samples);

% get final reward (note: this works only after the samples have been set!).
r = qrsim.reward();
```

1.4 Nomenclature

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[23]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.

1.5 Person Classifier Model

1.6 Concentration Models

To interpret the following concentration 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 plume 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:

$$x' = x\cos(\alpha_w) \tag{1.6}$$
$$y' = y\sin(\alpha_w). \tag{1.7}$$

1.6.1 Single Source Gaussian Concentration Model

$$c(x,y,z) = \exp\left(-\frac{1}{2}\begin{bmatrix} x - X_s \\ y - Y_s \\ z - H_s \end{bmatrix}^T \mathbf{\Sigma}^{-1} \begin{bmatrix} x - X_s \\ y - Y_s \\ z - H_s \end{bmatrix}\right) \tag{1.8}$$

1.6.2 Single Source Gaussian Dispersion Model

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

$$\begin{aligned} c(x', y', z) =& \frac{Q}{2\pi u a (x' - X'_s)^b} \exp\left(-\frac{(y' - Y'_s)^2}{2a(x' - X'_s)^b}\right) \\ & \left[\exp\left(-\frac{(z - H_s)^2}{2a(x' - X'_s)^b}\right) + \exp\left(-\frac{(z + H_s)^2}{2a(x' - X'_s)^b}\right)\right]. \end{aligned} \tag{1.9}$$

1.6.3 Multiple Sources Gaussian Dispersion Model

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

$$c(x', y', z) = \sum_{s=1}^{S} c(x', y', z; X'_s, Y'_s, H_s, Q_s). \tag{1.10}$$

1.6.4 Single Source Gaussian Puff Dispersion Model

FIXME: t random variable

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

$$
\begin{aligned}
c(x', y', z, t) = \sum_{i=1}^{I} \Bigg\{ & \frac{Q_s^i}{8(\pi a(x' - X_s')^b)^{3/2}} \\
& \exp\left(-\frac{(x' - X_s' - u(t - T_s^i))^2 + (y' - Y_s')^2}{2a(x' - X_s')^b} \right) \\
& \left[\exp\left(-\frac{(z - H_s)^2}{2a(x' - X_s')^b} \right) + \exp\left(-\frac{(z + H_s)^2}{2a(x' - X_s')^b} \right) \right] \Bigg\}.
\end{aligned}
\tag{1.11}
$$

1.6.5 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:

$$
c(x', y', z, t) = \sum_{s=1}^{S} c(x', y', z, t; X_s', Y_s', H_s, Q_s^{1..I_s}). \tag{1.12}
$$