MAS Project

Distributed Tasking Algorithm

User Manual

Version 1.0

MAS Project - Distributed Tasking Algorithm

Printed: 2014 Temasek Lab, NUS Singapore

Special thanks to:

All the people who contributed to this document.

Managing Editor

Dr. Feng Lin

Technical Editors

Dr. Dymkou Siarhei

Cover Editors

Zubik Aliaksandr

Team Coordinator

Dr. Teo Swee Huat Rodney

Production

Control Science Group, TL, NUS

Table of Contents

Part I Design Guide of the Distributed Tasking Algorithm 6

1 Problem Class Description ... 7
2 Solution Description ... 9
3 Algorithm Input/Output and Parameters to be set ... 9
4 Algorithm Procedure ... 10
Bundle construction phase ... 10
Consensus phase ... 11
5 Algorithm Properties ... 11
6 Example ... 12
7 Example 2 ... 16
8 Open research questions ... 18

Part II Manual to start control of UAVs in UDK 20

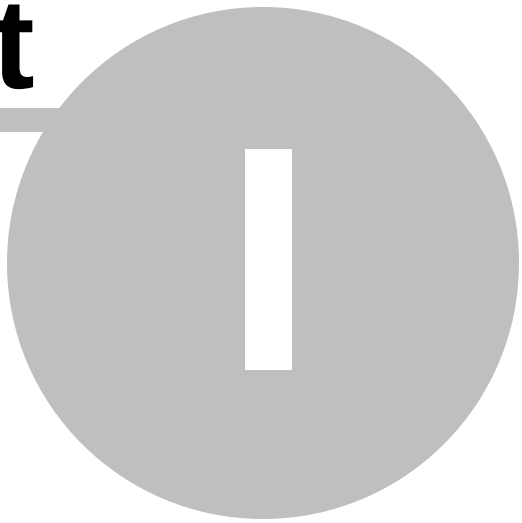
1 Install UDK ... 26
2 Choose the map ... 26
3 Run start.bat ... 26
4 Run algorithm ... 26
5 Key Matlab functions used in the demo ... 27
6 How to Change the variables ... 28

Index 0

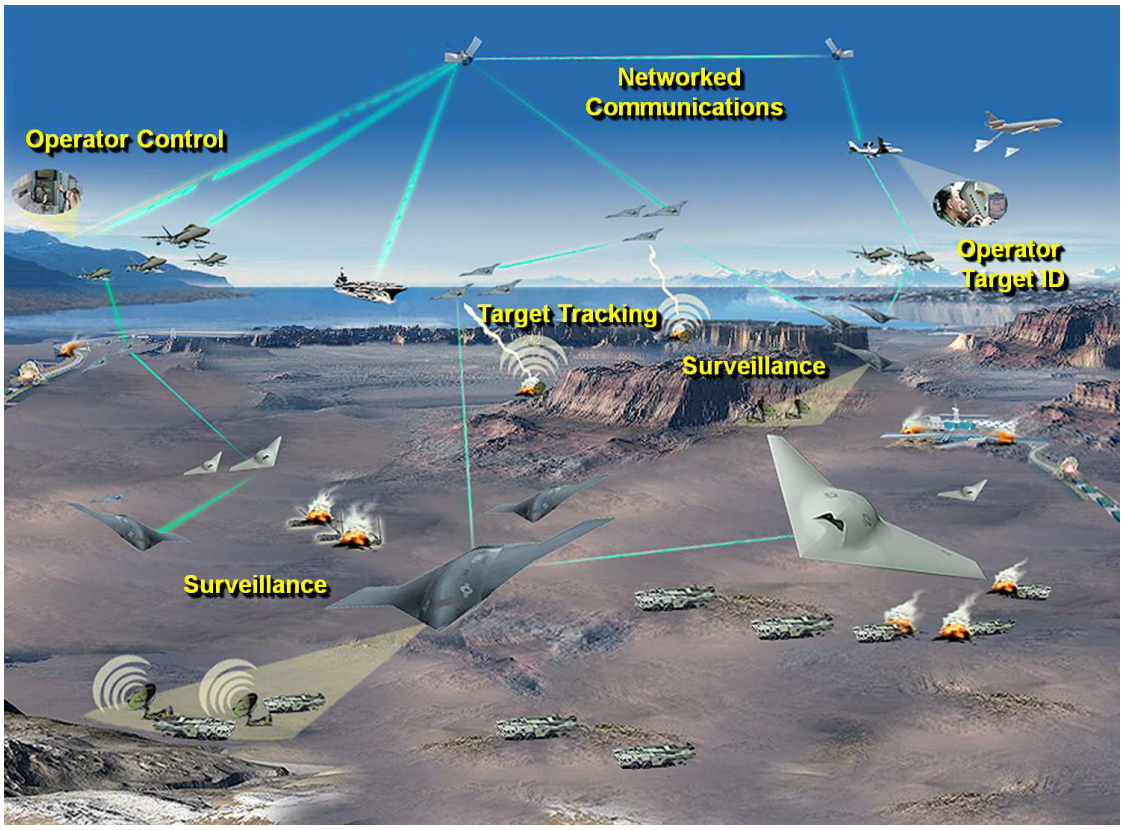
Introduction

This software package implements the Tasking Algorithm, a decentralized market-based protocol that provides provably good feasible solutions for multi-agent multi-task allocation problems over networks of heterogeneous agents. The current version supports tasks with time windows of validity, heterogeneous agent-task compatibility requirements, and score functions that balance task reward and fuel costs.

Part



1 Design Guide of the Distributed Tasking Algorithm



Network centric operations involve large teams of agents, with heterogeneous capabilities, interacting together to perform missions. These missions involve executing several different tasks such as conducting reconnaissance, surveillance, target classification, and rescue operations. Within the heterogeneous team, some specialized agents are better suited to handle certain types of tasks than others. For example, UAVs equipped with video can be used to perform search, surveillance and reconnaissance operations, human operators can be used for classification tasks, ground teams can be deployed to perform rescue operations, etc. The figure illustrates an example of such complex mission scenario involving numerous networked agents performing a variety of search, track, and surveillance tasks.

Ensuring proper coordination and collaboration between agents in the team is crucial to efficient and successful mission execution

Key question

- to coordinate team behavior to improve mission performance
- to hedge against uncertainty in dynamic environments
- to handle varying communication constraints

1.1 Problem Class Description

Item	Description
General description	***To assign UAVs to perform as many simultaneous service request as possible***
Starting conditions	**Given:** • The MAS is performing some job • Multiple requests for service with the following information: *Number of UAVs required* *Location where UAVs need to visit* *Earliest time of 1st visit* *Latest time of 1st visit* *Minimum duration per visit* *Maximum interval between visits*
Mission objective	**Find:** • UAVs to be assignet to request and the corresponding path to take to servise request

Item	Description
	• Variations to request with minimal change if the request cannot be met **That:** • Maximizes the number of service request that can be serviced
Constraints	**Subject to:** • UAVs perfomance and dynamics • Sensor perfomance • Air to Air datalink perfomance • LOS occlusion in area of operations • At least one UAV being directly connected to CSG for some time

The general system configuration for simulation research of distributed tasking problem are as follows:

Configuration

- UAV Airspace;
- Range for air-to-air communications between the UAVs;
- Homogenous\Heterogenous Service UAVs;
- Local information about targets;
- Static and moving targets.

Requirements

- ✓ Manage UAVs themselves automatically as a team in a hierarchical and / or distributed manner;
- ✓ Provide automatic self-allocation and self-deployment.;
- ✓ Maximize the local reward for each Service-UAVs.

Considerations

- ❖ Limited UAV sensing range and communication range;
- ❖ Breakdown of UAVs and variations of communication topologies;
- ❖ Distributed information fusion between UAVs;

❖ Distributed motion planning for UAVs.

1.2 Solution Description

The distributed tasking protocols with performance guarantees obtained through mathematical analysis and proofs as opposed to only via simulation and empirical tests (which are always limited).

Main issues: Coupling and Communication:

- Agent score functions depend on other agents decisions
- Joint constraints between multiple agents
- Agent optimization is based on local information

1.3 Algorithm Input/Output and Parameters to be set

Possible target (task) fields:	Possible UAVs fields
• **id** - *task id* • **type**- *task type* • **value** - *task reward* • **start**- *task start time (sec)* • **end**- *task expiry time(sec)* • **duration** - *task default duration* • **x**- *task position (meters)* • **y**- *task position (meters)* • **z**- *task position (meters)*	• **type** - *UAV type* • **avail**- *UAV availability (expected time in sec)* • **x**- *UAV position (meters)* • **y**- *UAV position (meters)* • **z**- *UAV position (meters)* • **velocity**- *UAV cruise velosity (m/s)* • **fuel** - *UAV fuel per meter*

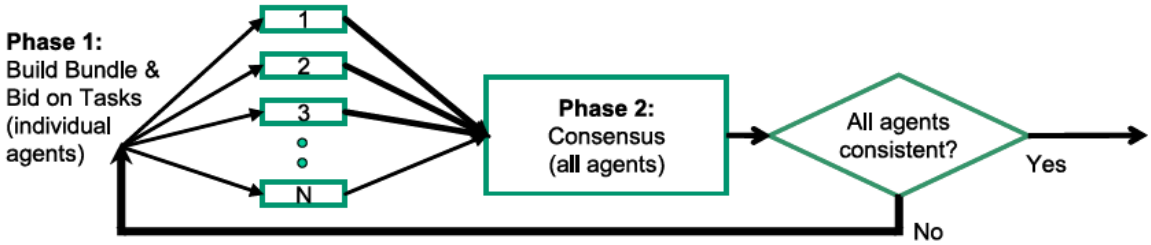
1.4 Algorithm Procedure

Consensus-Based Bundle Algorithm (CBBA) is a decentralized market-based protocol that provides provably good approximate solutions for multi-agent multi-task allocation problems over networks of heterogeneous agents. CBBA consists of iterations between two phases: a bundle building phase where each vehicle greedily generates an ordered bundle of tasks, and a consensus phase where conflicting assignments are identified and resolved through local communication between neighboring agents. There are several core features of CBBA that can be exploited to develop an efficient planning mechanism for heterogeneous teams.

- ***First, CBBA is a decentralized decision architecture, which is a necessity for planning over large teams due to the increasing communication and computation overhead required for centralized planning with a large number of agents.***

- ***Second, CBBA is a polynomial-time algorithm leading to a framework that scales well with the size of the network and/or the number of tasks (or equivalently, the length of the planning horizon).***

- ***Third, CBBA is capable of handling various design objectives, nonlinear agent models, and constraints, and under a few assumptions on the schoring structure, a provably good feasible solution guaranteed.***



1.4.1 Bundle construction phase

In contrast to the combinatorial algorithms, which enumerate all possible bundles for bidding, in CBBA each UAV creates just its single bundle which is updated as the assignment process progresses. During this phase of the algorithm, each UAV continuously adds tasks to its bundle in a sequential greedy fashion until it is incapable of adding any others. Tasks in the bundle are ordered based on which ones were added first in sequence, while those in the path are ordered based on their predicted execution times.

The bundle construction process is as follows: for each available task not currently in the bundle, or equivalently not in the path, the UAV computes a score for the corresponding task. The score is checked against the current winning bids, and is kept if it is greater. Out of the remaining ones, the UAV selects the task with the highest score and adds that task to its bundle.

Computing the score for a task is a complex process which is dependent on the tasks already in the agents path (and/or bundle). Selecting the best score for task can be performed using the following

two steps. First, task is "inserted" in the path at some location (i.e. we are change old path and construct the new one) . The score for each task is dependent on the time at which it is executed,

motivating the second step, which consists of finding the optimal execution time given the new path. This can be found by solving continuous time optimization problem. The constraints state that the insertion of the new task into path cannot impact the current times (and corresponding scores) for the tasks already in the path.
Once the scores for all possible tasks are computed, the scores need to be checked against the winning bid list, to see if any other agent has a higher bid for the task. And the final step is to select the highest scoring task to add to the bundle. The bundle, path, times, winning agents and winning bids vectors are then updated to include the new task. The bundle building recursion continues until either the bundle is full (i.e. the limit is reached), or no tasks can be added for which the agent is not outbid by some other agent.

1.4.2 Consensus phase

Once agents have built their bundles of desired tasks they need to communicate with each other to resolve conflicting assignments amongst the team. After receiving information from neighboring agents about the winning agents and corresponding winning bids, each agent can determine if it has been outbid for any task in its bundle. Since the bundle building recursion, described in the previous section, depends at each iteration upon the tasks in the bundle up to that point, if an agent is outbid for a task, it must release it and all subsequent tasks from its bundle. If the subsequent tasks are not released, then the current best scores computed for those tasks would be overly conservative, possibly leading to a degradation in performance. It is better, therefore, to release all tasks after the outbid task and redo the bundle building recursion process to add these tasks (or possibly better ones) back into the bundle.

This consensus phase assumes that each pair of neighboring agents synchronously shares the following information vectors:

- *the winning UAVs list,*
- *the winning bids list,*
- *and the vector of timestamps representing the time stamps of the last information updates received about all the other agents.*

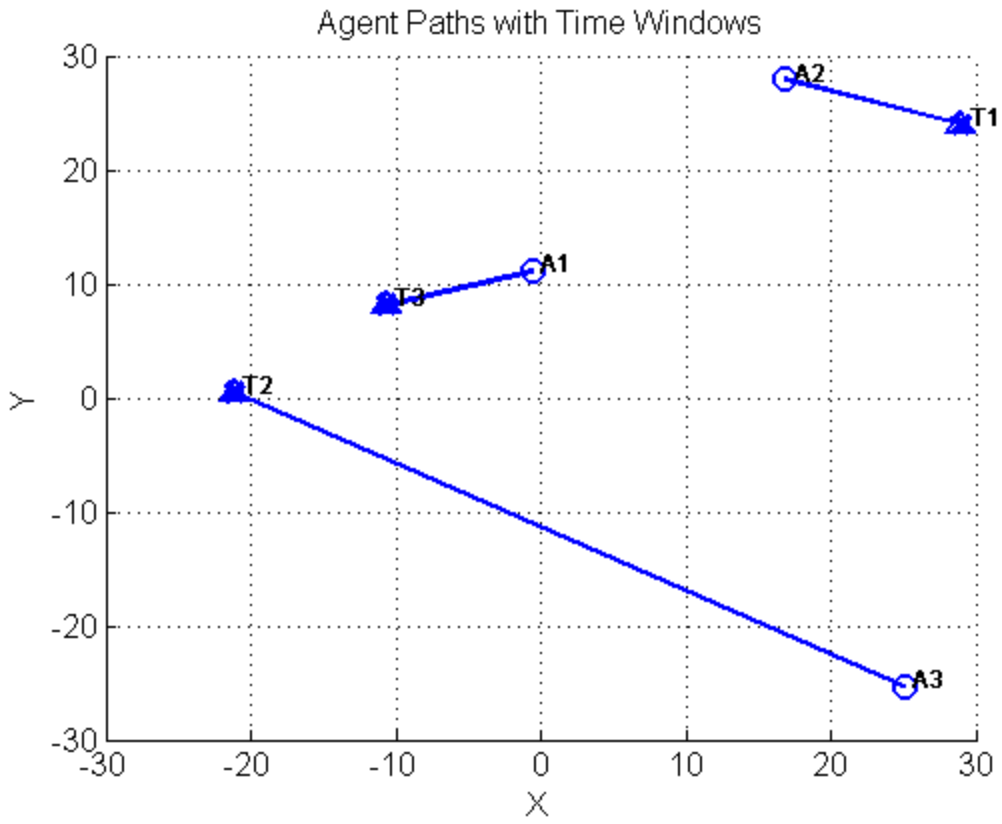
And then using some communication protocols and "design rules" the algorithm will assign the best UAV to perform the corresponding task. From here, the algorithm returns to the first phase where new tasks can be added to the bundle. CBBA will iterates between these two phases until no changes to the information vectors occur anymore.

1.5 Algorithm Properties

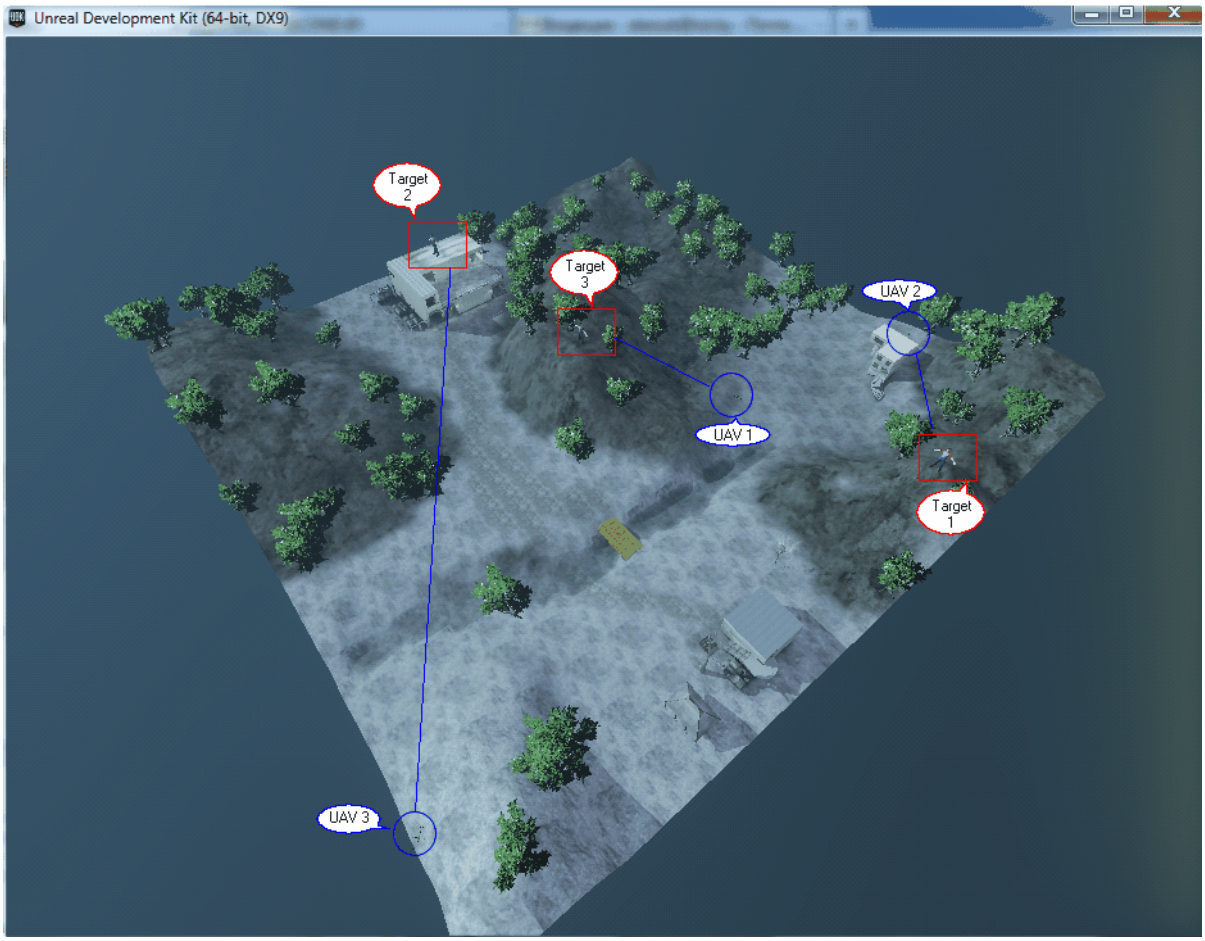
- Task selection - Polynomial-time, provably good feasible solutions
- Guaranteed real-time convergence even with inconsistent environment knowledge
- Time-varying score functions (e.g. time-windows of validity for tasks)

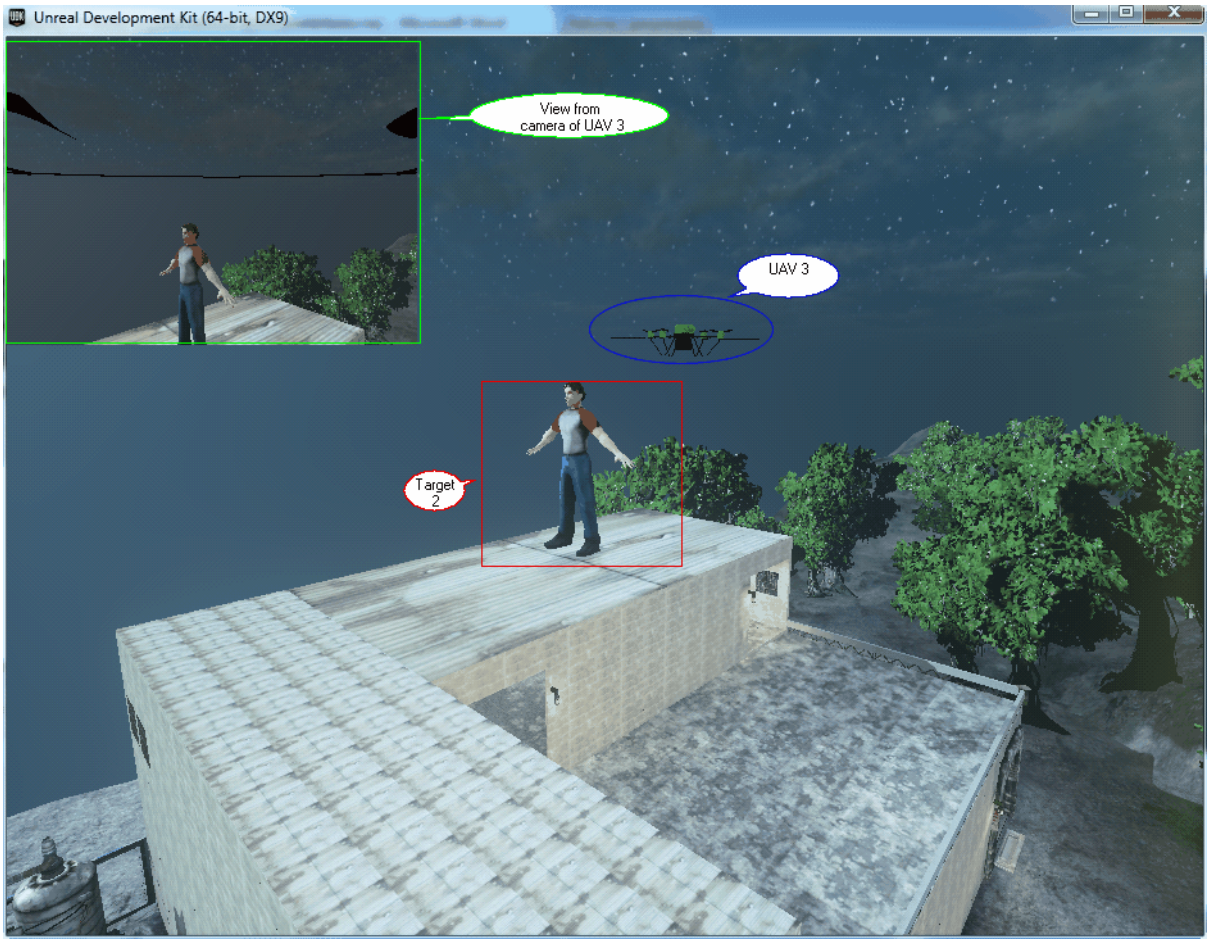
1.6 Example

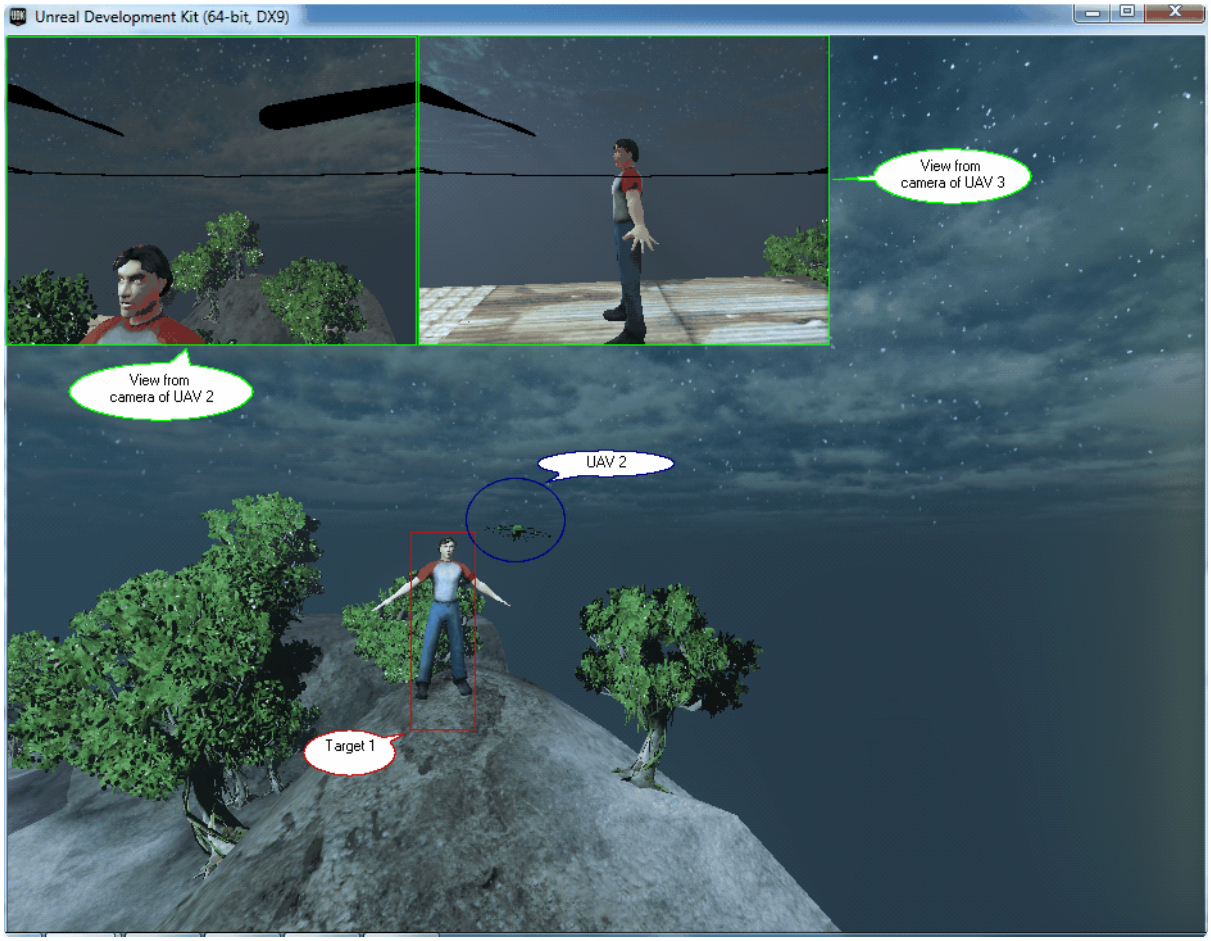
For example the so called Tethered UAVs Self-Assignment Problem is a particular case of the problem described above. And the solution of this particular problem presented in a figure below, namely, we find the logic that enabled the tethered UAVs (3 UAVs) to self-deploy one UAV to each specified location (3 locations).



Start position:





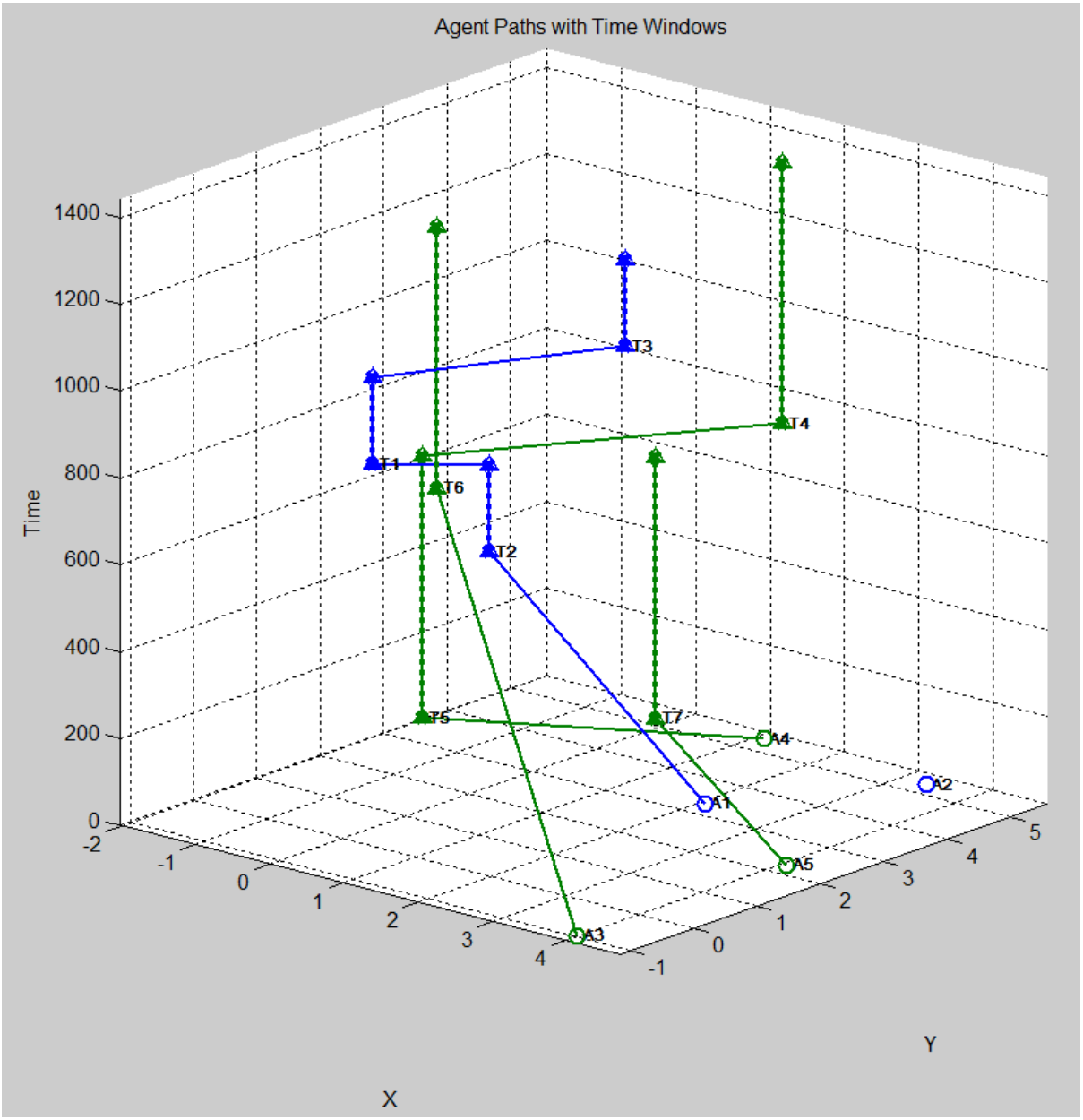


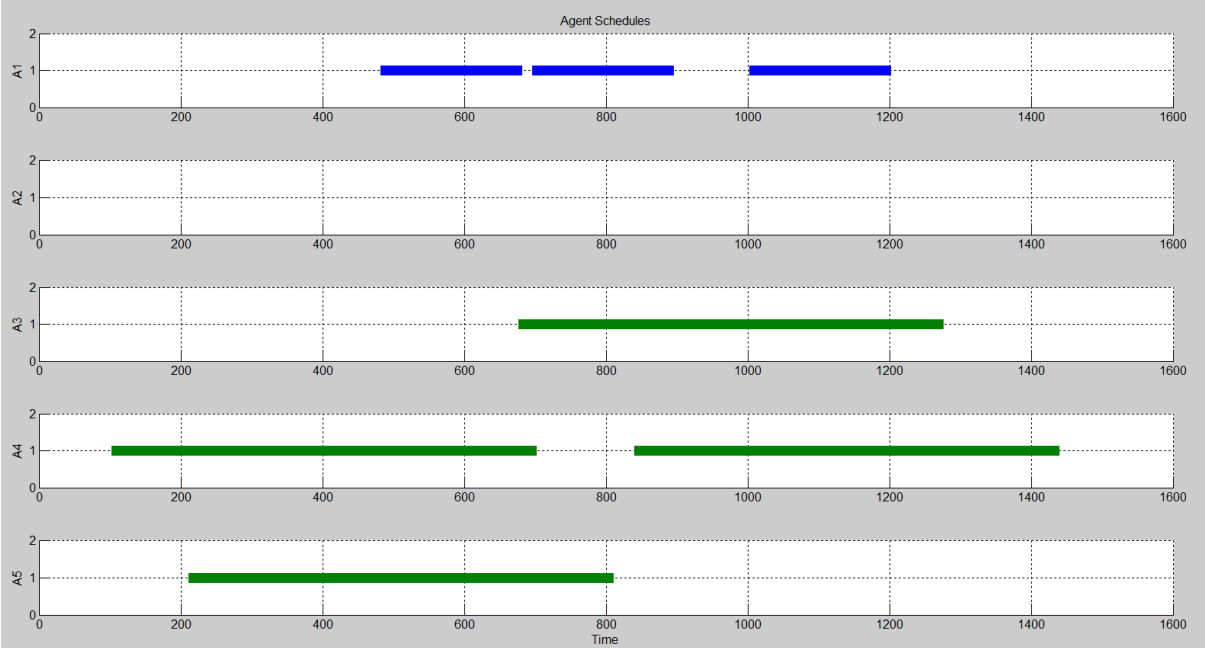
1.7 Example 2

Tasks with time windows of validity, heterogeneous agent-task compatibility requirements, and score functions that balance task reward and fuel costs.
In this example we have 2 types of UAVs and 2 types of tasks, namely:

- A1 and A2 are the UAVs of type 1 (blue circles);
- A3-A5 are the UAVs of type 2 (green circles);

❖T1-T3 the tasks of type 1 (blue triangles),
❖T4-T7 the tasks of type 2 (green trianlges).





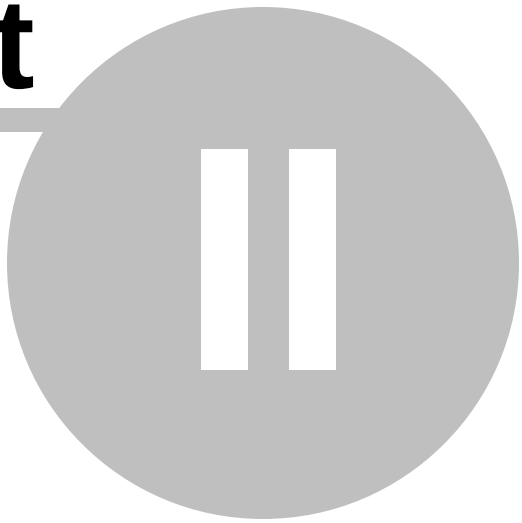
1.8 Open research questions

1) Possibility to consider the case when the current missions are in progress and need to recompute when new tasks and new assets are added/removed or the case when CGS is available again.

2) Possibility to consider task allocation in order to ensure flyable paths, for example to minimise turn radius.

3) Consider fully nonlinear cases. (Path, cost function and etc.)

Part



2 Manual to start control of UAVs in UDK

A control program for UAVs is implemented based on the new Unreal Development Kit version of USARSim in order to create a simulation environment for UAVs. The purpose of the implemented software is to be helpful in research, development and testing of control algorithms for quadrocopters operating in a three dimensional environment as well as in research of learning algorithms for autonomous quadrocopter behavior. The control program is supposed to act as a framework and a foundation which can easily be extended with new algorithms that can then be tested for functionality in the simulated environment without any risk for actual hardware. Furthermore the implemented software should allow users to focus on the implementation of their algorithms, releasing them of the task to create whole new control programs and deal with low level programming of underlying software layers every time a new algorithm is to be evaluated.

Simulation

Simulation makes it possible to test control algorithms and system behaviors of machines without the need for a lot of expenses for the actual hardware that is simulated. While a real system does only exist once, a system in a simulation can be reproduced several times if needed, allowing for economies in otherwise particularly expensive areas such as multi-robot control. Furthermore a lot of simulations can be run in parallel if the needed processing power is available, possibly speeding up research and development processes. On top of this simulated hardware does not need to be repaired, maintained or stored. A crash of a prototype system in real life could mean hours of repairs during which the system is unavailable for further tests and development, resulting in high costs. A crash in a simulation though needs only a hand full of mouse clicks and the simulation can be started anew. This can be especially advantageous in the case of machine learning where new algorithms might not always act as expected or where they are expected to produce crashes in the beginning before converting more or less slowly to a desired behavior. In a simulation a machine can learn the basics of how to behave and to get along well within its environment. After that the algorithms developed in the simulation can be applied to a real machine letting it learn the remaining details in the real world. In some simulations it is also possible to adjust the speed of time and thus let a system learn much faster than it ever could in a real situation.

USARSim

USARSim is an acronym for Unifed System for Automation and Robot Simulation. It was designed as an open source high-fidelity simulation of robots and environments based on the Unreal Tournament game engine. It’s intended as a general purpose research tool with applications ranging from human computer interfaces to behavior generation for groups of heterogeneous robots. In addition to research applications, USARSim is the basis for the RoboCup rescue virtual robot competition as well as the IEEE Virtual Manufacturing Automation

Competition (VMAC).

USARSim loads off the most diffcult parts of simulation to the Unreal game engine, so that the developers behind the project and users can concentrate more on the robot-specific tasks of modeling platforms, control systems, sensors, interface tools and environments. The 3D rendering and physics calculations are all handed by the underlying Unreal Engine. USARSim itself provides several legged and wheeled robots, aerial robots and even submarine robots, as well as a battery of sensors and actuators and virtual environments, i.e. maps, that the robots can be placed in.

The sensors provided by USARSim can be generally divided into three categories. First there are proprioceptive sensors, including battery state and headlight state, second position estimation sensors, including location, rotation, and velocity sensors and third there are perception sensors, including sonar, laser, sound, and pan-tilt-zoom cameras. USARSim defines a hierarchical architecture for sensor models as well as for robot models. A sensor class defines a type of sensor and every sensor is defined by a set of attributes stored in a configuration file. Perception sensors, for example, are commonly specified by range, resolution, and field of view.

Beyond that USARSim provides users with the capability to build their own sensors and robots.

USARSim is available in an older version for Unreal Tournament 2004, in an Unreal Tournament 3 version and is right now in the process of being ported from the Unreal Tournament 3 system to the newer free Unreal Development Kit. Since this process is not finished yet and the UT3 version of USARSim is not compatible with the UDK version, the UDK version of USARSim does not yet support all Robots, Maps and Sensors the previous versions of USARSim supported and models, maps and sensors from the previous version of USARSim are usually not usable with the new UDK version.

USARSim provides a socket based interface which is based on Gamebots, a modi?cation for Unreal Tournament, through which it is possible to communicate with the simulated robots directly, bypassing the Unreal Client. This also enables controller applications to reside on different computers than the Unreal Engine thus allowing to control a virtual robot over a network connection. A sketch of this architecture can be seen in figure 1. A brief overview of Gamebots is provided in. The communication protocol implemented by USARSim is based on simple text messages being sent between the controller application and USARSim. All this enables users to develop and test their own control programs and user interfaces without limitations in programming language and operating system.

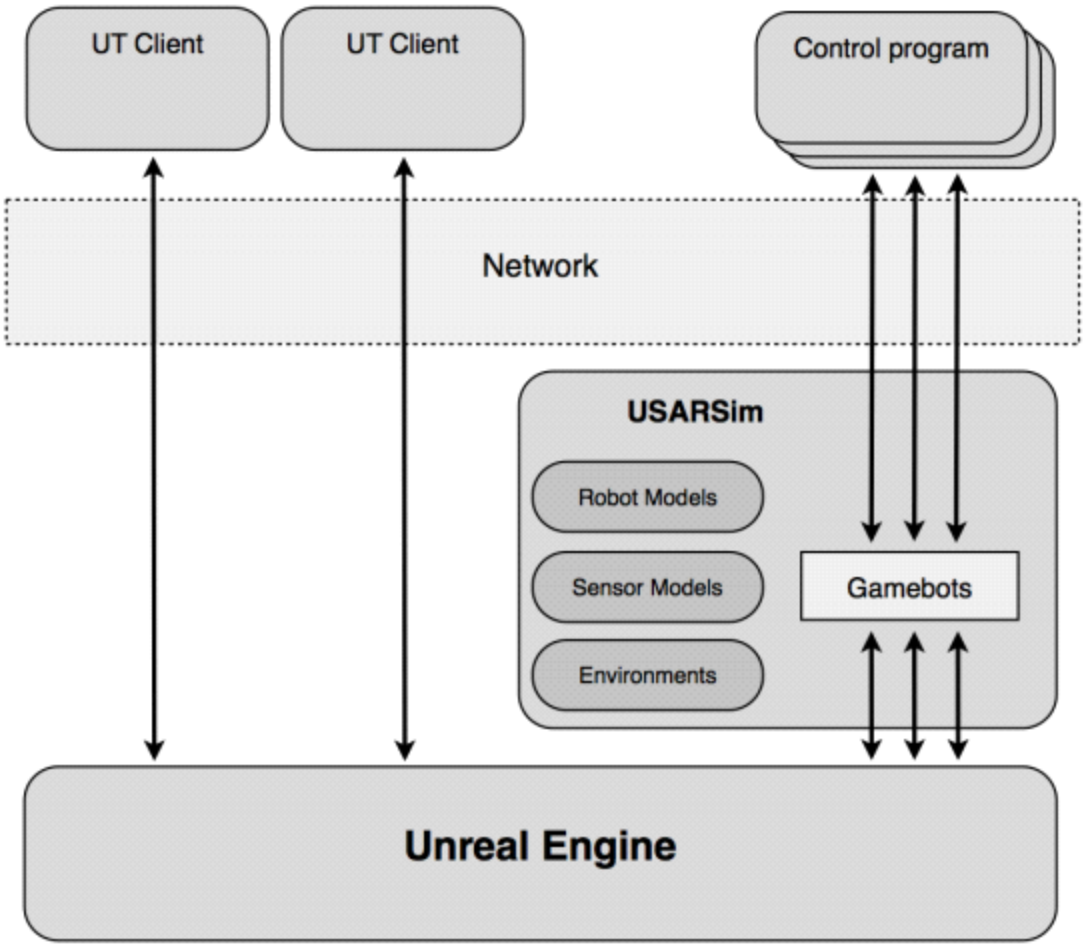


Figure 1.1: USARSim architecture

USARSim is an open source project licensed under the Gnu Public Licence and is freely available on the internet.

UAV (AirRobot)

The AirRobot is a four-rotor electrical helicopter with flight stabilization control produced by AirRobot Co. (http://www.airrobot.com/englisch/index.php). The AirRobot serves many purposes including, but not limited to, exploration, observation, documentation, and measurement.

In USARSim, we use classname USARBot.AirRobot to represent this robot.

Figure 1.2: Real AirRobot

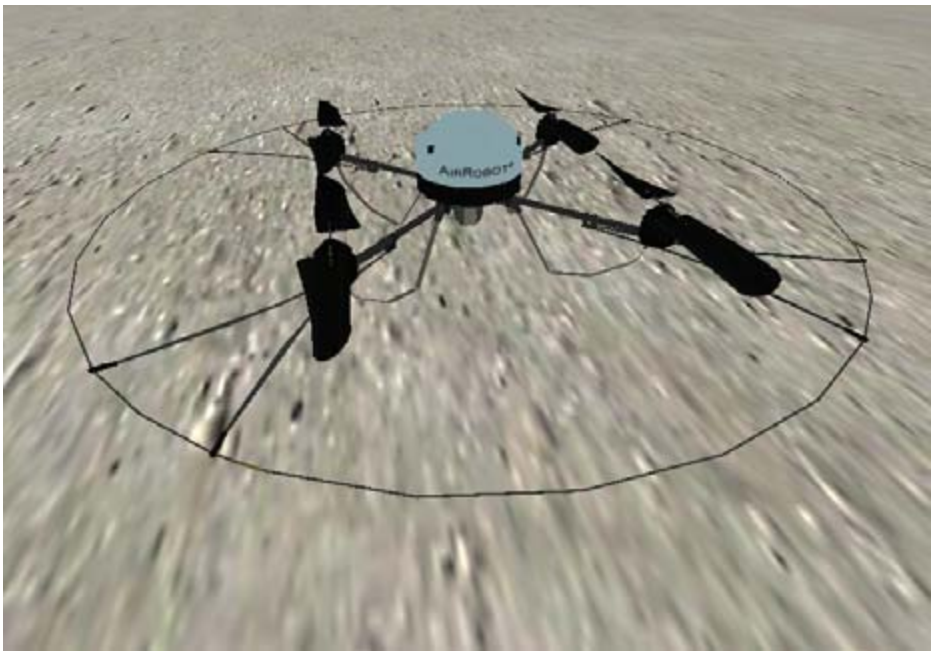


Figure 1.3: Simulated AirRobot

In summary, an AirRobot has:

- Four propellers

- One color camera that can tilt
- Weight: 1 kg
- Payload: 200 g

In USARSim, the AirRobot is equipped with

- One “tilt-only” color camera

The AirRobot specification is as follows:

- Dimension: Length x Width x Height = 0.999m x 0.999m x 0.194m
- Maximum altitude velocity: 5 m/s
- Maximum linear : 1.5708 rad/s

Control Program

Overview

The control program is generally responsible for steering a quadrocopter in the three dimensional simulation environment created by the Unreal Engine and made accessible via USARSim. The program first requires the user to enter some basic data considering the server properties, used quadrocopter model and missions for the UAV to accomplish. The program will create a connection to the communication socket of USARSim via which the control program will communicate with the UAV in the simulation. The UAV will send its sensor data and status data over the link created to the control program which then based on this data will decide what action the UAV is supposed to take and sends the appropriate commands over the communication link back to the UAV.

Because the control program is based on a client-server architecture it is not restricted to controlling simulated quadrocopters only. Given a real quadrocopter implemented the USARSim communication protocol the presented control program could receive sensor readings from and send commands to the quadrocopter and thus control the quadrocopter in a real environment. Further on, since the underlying Unreal Engine allows for multiple connections at the same time, it is possible to simultaneously have several quadrocopters in the same simulated environment, each of which is controlled by another instance of the control program. This allows for multi-UAV scenarios to be tested and evaluated.

Architecture

As stated before the whole simulation environment is based on the Unreal Engine which is responsible for generating the graphics and calculating the physics of the three dimensional virtual environment. On top of this builds USARSim which provides models of robots as well as sensors and their functionalities, a number of standardized environments

and a communication interface to use for robot control. The control program then is built on top of this basis, as can be seen in figure 1.3 and takes responsibility for the interaction between user and simulation environment and robot control.

To provide an easy to understand and easy to modify structure the control program is divided into several modules. Each of this modules has its own functionality and purpose. The modules can be categorized into three main parts in a model view-controller like fashion. First there is the interface to the user which contains the main window and any dialog windows appearing in the program. These are responsible for gathering needed information, reacting to user requests and providing the user with information about the UAV that is being controlled. Second there is the part that is responsible for the programs functionality and models the different aspects of it. This part contains components like for example a client which handles the communication link between control program and USARSim or a UAV component encapsulating all the different aspects of a UAV's functionality and others. The third part is a control layer responsible for coordinating the different components among each other and with the user interface. The basic structure and communication between components can be seen in ?gure 1.3.

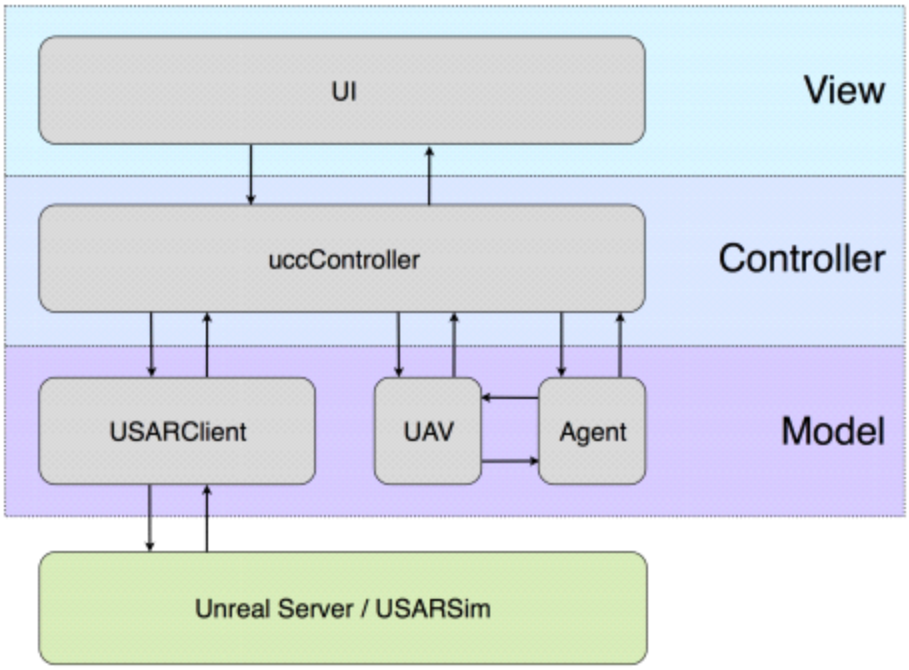


Figure 1.3: Control program architecture overview

2.1 Install UDK.

Installation instructions

Please see the readme file included in the release for the most up-to-date instructions.

1. Download and Install the latest UDK (http://www.udk.com/, the last checked version is from Feb-2013). Install this to a directory named UDK\UDK-yyyy-mm. In this case "yyyy" is the year of your UDK release and "mm" is the month.
2. Using a client such as Git Gui, open a bash window in the same directory that you specified in step 1 (UDK-yyyy-mm) and type:

   git clone git://git.code.sf.net/p/usarsim/code

When you have a sourceforge account which is added to the contributors list you have read/write access. In that case type:

git clone ssh://yourUserName@git.code.sf.net/p/usarsim/code

1. Move all of the files (including the .git folder) from the usarsim folder into the directory specified in step 1. Note that the .git folder is only visible when the Windows Folder View option "Show hidden files and directories" is active.
2. To compile USARSim, run "make" in the UDK-yyyy-mm folder.

2.2 Choose the map

Running

1. Execute make.bat (might require Administrative privileges to run correctly)
2. Start usarsim using one of the map bat files located in USARRunMaps or your user map.

2.3 Run start.bat

This file will start the demonstration in UDK enviroment and the information about the UAVs reading from the corresponding .mat files which are generated by Matlab code.

2.4 Run algorithm

Run the MainTestScript.m from the root directory. The algorithm will generate automatically the .mat files for each UAVs, which are contains the information about corresponding schedules and path.

2.5 Key Matlab functions used in the demo

1) MainTestScript.m

Description: *Initializes problem and calls CBBA.*

- Initialize global variables;
- Initialize possible agent fields;
- Initialize possible task fields;
- Define scenario;
- Call CBBA.

2) CBBA_Init.m

Description: *Initialize CBBA Parameters*

- Add specialized agent types and task types;
- Set agent-task pairs (which types of agents can do which types of tasks);
- Set agent maximum bundle depth.

3) CBBA_Main.m

Description: *Contains main CBBA Function*

Main CBBA loop (runs until convergence):
1. Communicate (Perform consensus on winning agents and bid values (synchronous))
2. Run CBBA bundle building/updating (Run CBBA on each agent (decentralized but synchronous))
3. Convergence Check (Determine if the assignment is over, if not maintain loop);
4. Map path and bundle values to actual task indices;
5. Compute the total score of the CBBA assignment.

4) CBBA_Communicate.m

Description: *Runs consensus between neighbors and checks for conflicts and resolves among agents.*

5) CBBA_Bundle.m

Description: *Main CBBA bundle building/updating (runs on each individual agent)*

- Update bundles after messaging to drop tasks that are outbid;
- Bid on new tasks and add them to the bundle

6) CBBA_BundleRemove.m

Description: *Update bundles after messaging to drop tasks that are outbid*

7) CBBA_BundleAdd.m

Description:

- *Create bundles for each agent;*
- *Bid on new tasks and add them to the bundle*

8) Scoring_CalcScore.m

Description: *Calculates score of doing a task and returns the expected start time for the task.*

9) PlotAssignmennts.m

Description: *Plot CBBA output*

2.6 How to Change the variables

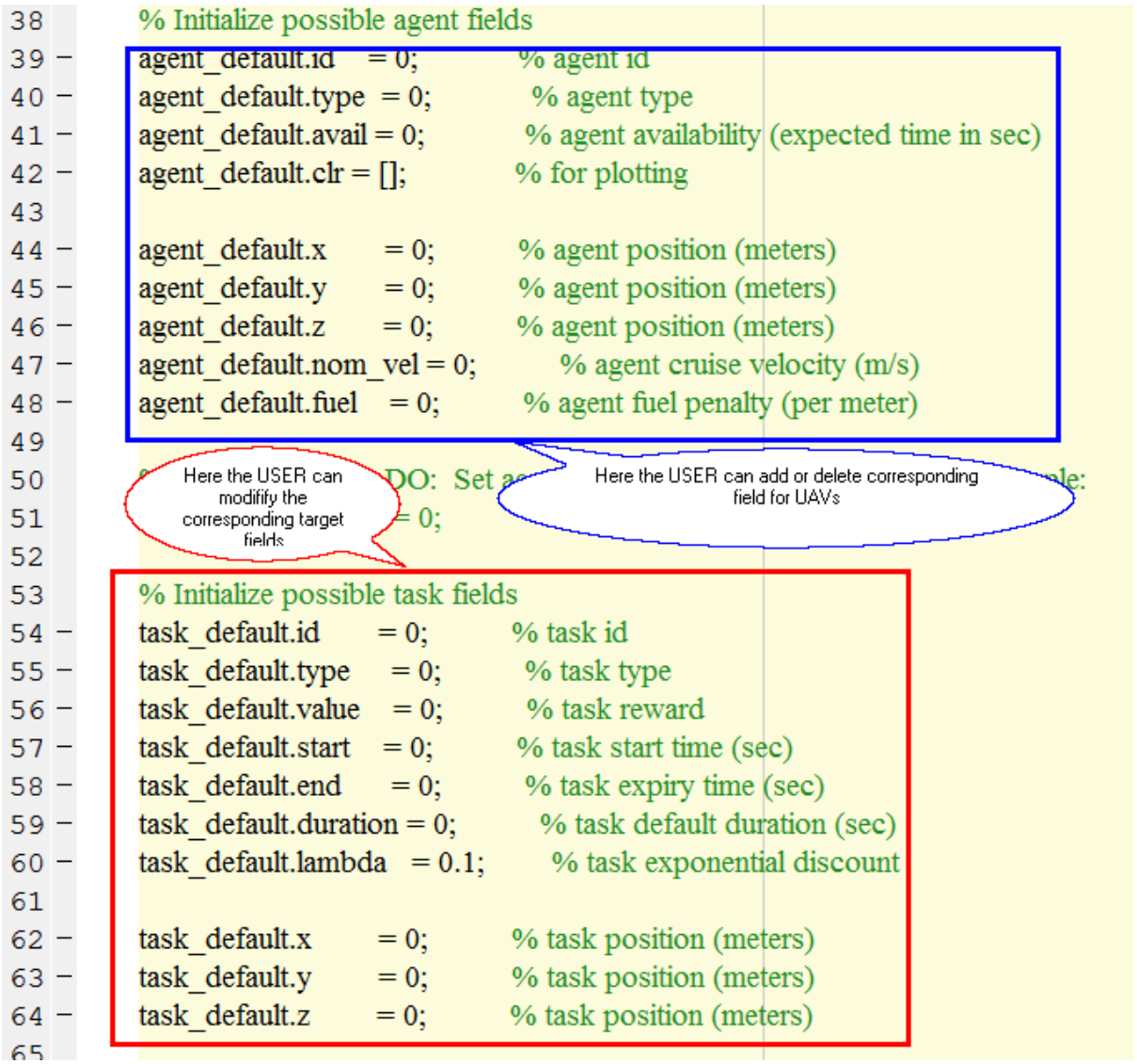
The USER should open the file MainTestScript.m
and he can modify the following parameters:

1. Map size

```
16      %-------------------------------------------------------------------%
17      % Initialize global variables
18      %-------------------------------------------------------------------%
19
20 -    WORLD.CLR  = rand(100,3);
21
22 -    WORLD.XMIN = -2.0;
23 -    WORLD.XMAX =  2.5;
24 -    WORLD.YMIN = -1.5;
25 -    WORLD.YMAX =  5.5;
26 -    WORLD.ZMIN =  0.0;
27 -    WORLD.ZMAX =  2.0;
28 -    WORLD.MAX_DISTANCE = sqrt((WORLD.XMAX - WORLD.XMIN)^2 + ...
29                          (WORLD.YMAX - WORLD.YMIN)^2 + ...
30                          (WORLD.ZMAX - WORLD.ZMIN)^2);
31
```

Here the User can set his own map

2. Define own parameters for the UAVs and targets (tasks)



```
71
72      % QUAD
73 -    agent_quad         = agent_default;
74 -    agent_quad.type    = CBBA_Params.AGENT_TYPES.QUAD; % agent type
75 -    agent_quad.nom_vel = 2;        % agent cruise velocity (m/s)
76 -    agent_quad.fuel    = 1;        % agent fuel penalty (per meter)
77
78      % CAR
79 -    agent_car          = agent_default;
80 -    agent_car.type     = CBBA_Params.AGENT_TYPES.CAR;  % agent type
81 -    agent_car.nom_vel  = 2;        % agent cruise velocity (m/s)
82 -    agent_car.fuel     = 1;        % agent fuel penalty (per meter)
83
84      % Create some default tasks
85
86      % Track
87 -    task_track         = task_default;
88 -    task_track.type    = CBBA_Params.TASK_TYPES.TRACK;     % task type
89 -    task_track.value   = 100;   % task reward
90 -    task_track.start   = 0;     % task start time (sec)
91 -    task_track.end     = 3000;  % task expiry time (sec)
92 -    task_track.duration = 200;     % task default duration (sec)
93
94      % Rescue
95 -    task_rescue        = task_default;
96 -    task_rescue.type   = CBBA_Params.TASK_TYPES.RESCUE;     % task type
97 -    task_rescue.value  = 100;   % task reward
98 -    task_rescue.start  = 0;     % task start time (sec)
99 -    task_rescue.end    = 2000;  % task expiry time (sec)
.00 -    task_rescue.duration = 600;     % task default duration (sec)
.01
```

Here the USER can define his own types of robots

Here the USER can define his own types of targets (task)

3. Select the total number of UAVs and Tasks which will be simulated in the scenario

```
104     % Define sample scenario
105     %-------------------------------------------------------------%
106
107 -   N = 5;     % # of agents
108 -   M = 7;     % # of tasks
109
```

USER can change the total number of UAVs of all types

USER can change the total number of tasks

4. Define upper limit of the tasks which can be serviced by one UAV (file CBBA_Init.m)

```
10 -    CBBA_Params.MAX_DEPTH    = 3;          % maximum bundle depth (remember to set it for each scenario) (tip
11
12      % FOR USER TO DO: Add specialized age                    k types
13
14      % List agent types
15 -    CBBA_Params.AGENT_TYPES.QUAD   = 1;
16 -    CBBA_Params.AGENT_TYPES.CAR    = 2;
17
18      % List task types
19 -    CBBA_Params.TASK_TYPES.TRACK   = 1;
20 -    CBBA_Params.TASK_TYPES.RESCUE  = 2;
21
22      % Initialize Compatibility Matrix
23 -    CBBA_Params.CM = zeros(length(fieldnames(CBBA_Pa                    )), ...
24                  length(fieldnames(CBBA_Params.TASK_
25
26      % FOR USER TO DO: Set agent-task pairs (which types                  pes of tasks)
27 -    CBBA_Params.CM(CBBA_Params.AGENT_TYPES.QUAD, CBBA_Params.TASK_TYPES.TRACK)  = 1;
28 -    CBBA_Params.CM(CBBA_Params.AGENT_TYPES.CAR,  CBBA_Params.TASK_TYPES.RESCUE) = 1;
29
30 -    return
```

USER can define the maximum number of tasks that one UAV can service

Here the USER can define for each type of UAV the corresponding type of task that can be serviced by that UAV

5. ***Define own cost function in the file Scoring_CalcScore.m***