MASC Project

Formation control of fixed-wing UAVs
Target detection, image tracking and data fusion
Distributed tasking Algorithm
Distributed target search & Air-net coverage Algorithm

User Manual

Version 1.0

MASC Project

Printed: 2014 Temasek Lab, NUS Singapore

Special thanks to:

All the people who contributed to this document.

Managing Editor

Prof. Ben M. Chen

Technical Editors

Dr. Liao Fang

Dr. Lin Feng

Dr. Meng Wei

Dr. Dymkou Siarhei

Dr. Peng Kemao

Cover Editors

Dr. Dymkou Siarhei

Team Coordinator

Dr. Teo Swee Huat Rodney

Production

Control Science Group, TL, NUS

Table of Contents

Introduction

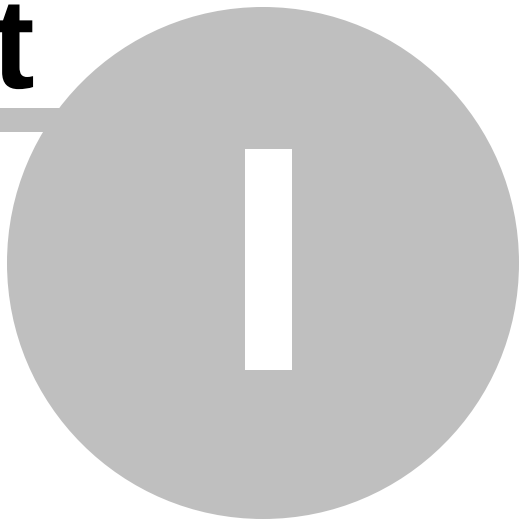
Formation mini-toolbox provides a simulation tool for formation control of fixed-wing UAVs. Namely it provides a convenient interface for users to set and modify parameters and run the simulation for formation control of a team of fixed-wing SB-III UAVs.

A software toolbox, called “VisDet”, has been developed to test the proposed target detection and image tracking algorithms using C++ language. This toolbox proposed vision algorithms, including target detection, image tracking and data fusion, have been packaged in a Class using C++ language. The algorithms have also been tested in an onboard computer of the UAV in real flight. Also it provide a user interface to visualize the processing results and evaluate the detection performance in real time.

Tasking software package implements the Distributed 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.

Coverage and search Mini-toolbox provides a simulation tool for distributed air-net communication coverage and distributed target search of multiple UAVs. These algorithms have been tested in Matlab and/or Unreal simulator. Search algorithm also has been written in C language for test. Detailed design guild for these two algorithms have been presented.

Part



1 Formation control of fixed-wing UAVs

1.1 About This Mini-Toolbox

This mini-toolbox provides a simulation tool for formation control of fixed-wing UAVs.

1.2 System Requirements

- Window XP and above
- Matlab 2008 above

1.3 Overview of This Mini-Toolbox

This mini-toolbox is an implementation of the formation control algorithm proposed in the D8 report. It provides a convenient interface for users to set and modify parameters and run the simulation for formation control of a team of fixed-wing SB-III UAVs.

In this mini-toolbox, the users may specify one of the UAVs as the leader and specify its yaw angle such that it can lead the team of UAVs to fly along a specified direction.

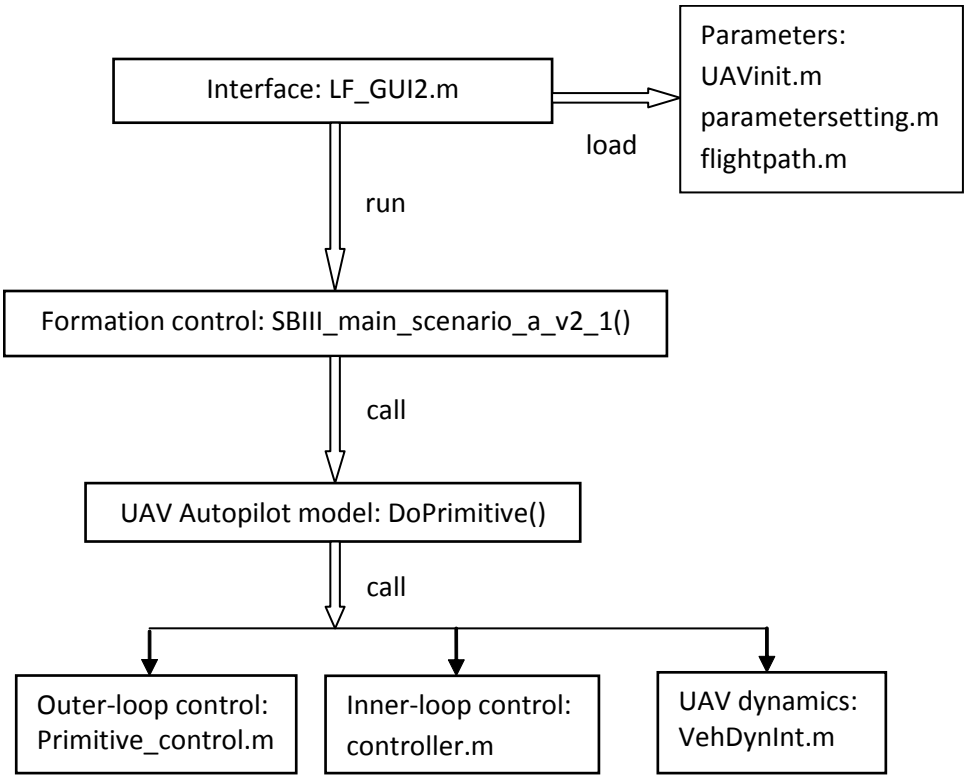
This mini-toolbox is compatible to the change of some parameters, which are listed as follows.

- Number of UAVs
- Simulation times
- Starting time of formation
- Iteration times
- Leader's identity number
- Maximum acceleration
- Maximum yaw rate
- Maximum velocity
- Minimum velocity

- epsa
- rhoa
- Ra
- Rc
- Spacing between UAVs
- formation control gain K_{f}
- formation control gain K_{v}
- formation shape
- disturbance
- Initial states and commands of UAVs
- flight path

The software consists of UAV model in Matlab file ``DoPrimitive.m'', formation control algorithm in Matlab file ``SBIII_main_scenario_a_v2_1.m'' and the interface in Matlab file ``LF_GUI2''.

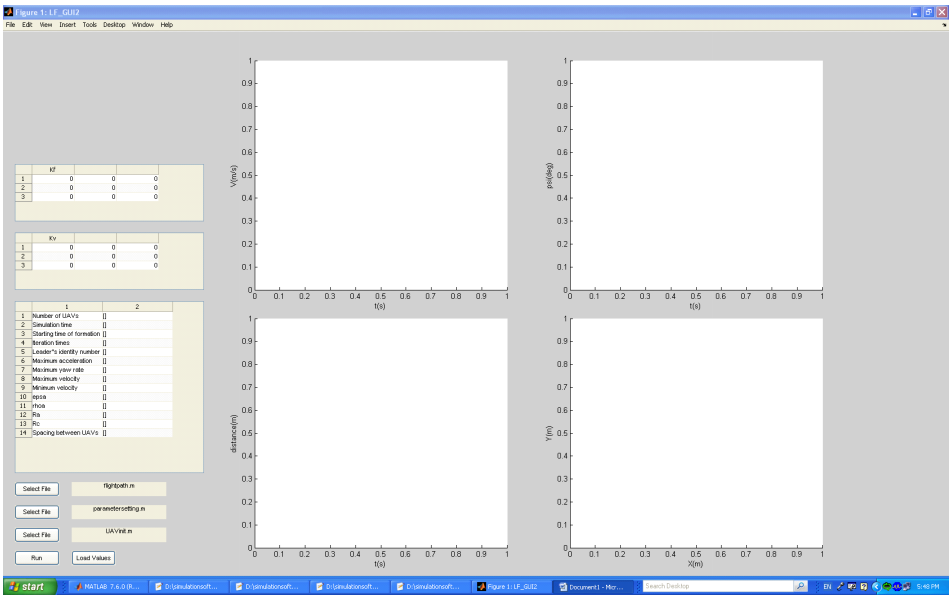
``LF_GUI2'' is the main program. It calls the subroutine ``SBIII_main_scenario_a_v2_1.m'', which calls ``DoPrimitive.m''. The software structure is shown as follows.



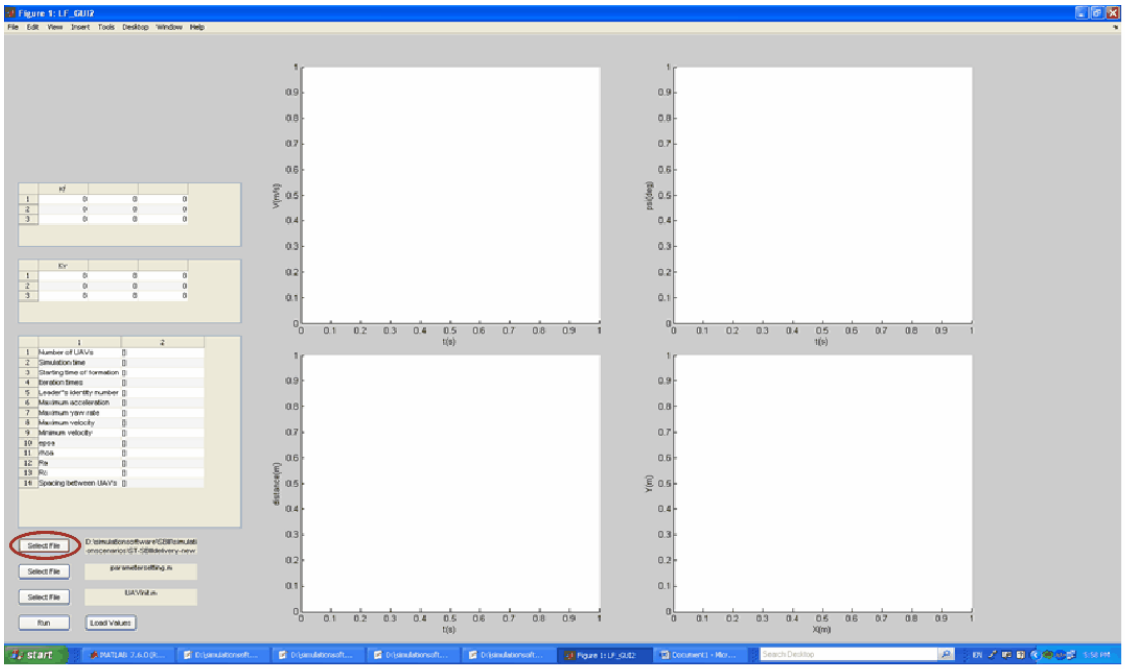
1.4 How to Use This Mini-tool Box

The procedures to use the mini-toolbox are given as follows.

1. Run the main program “LF_GUI2”. An interface shown in Figure 1 will pop up.

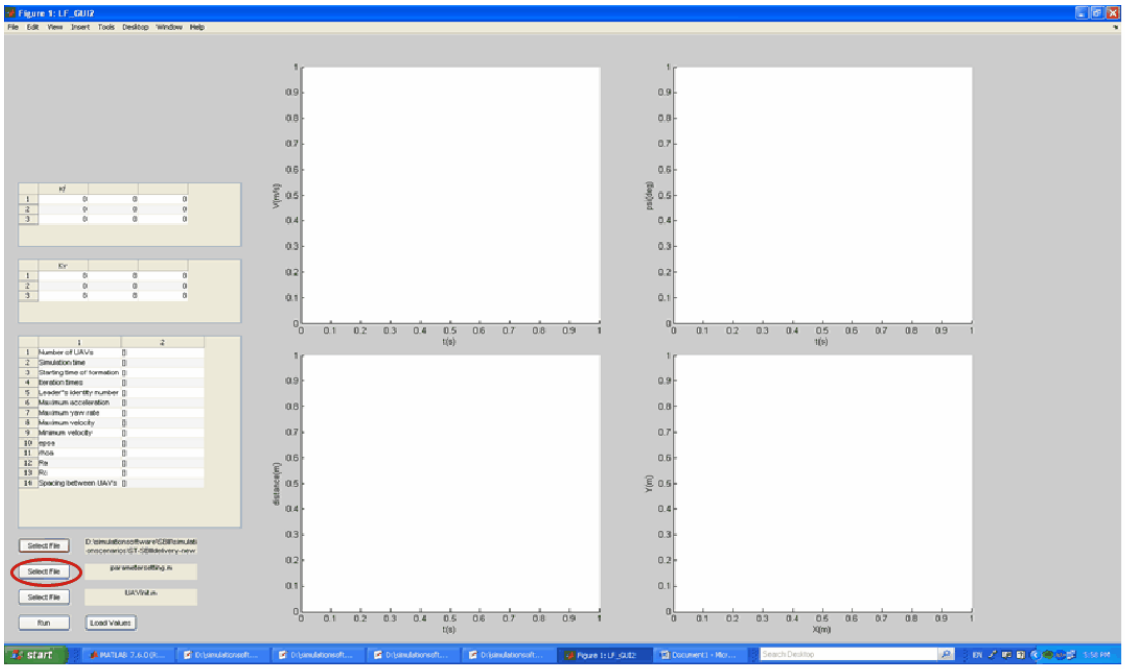


2. Click on the first “Select File” button to choose the file containing the specified flight path in current directory. The default file name is “flightpath.m”.

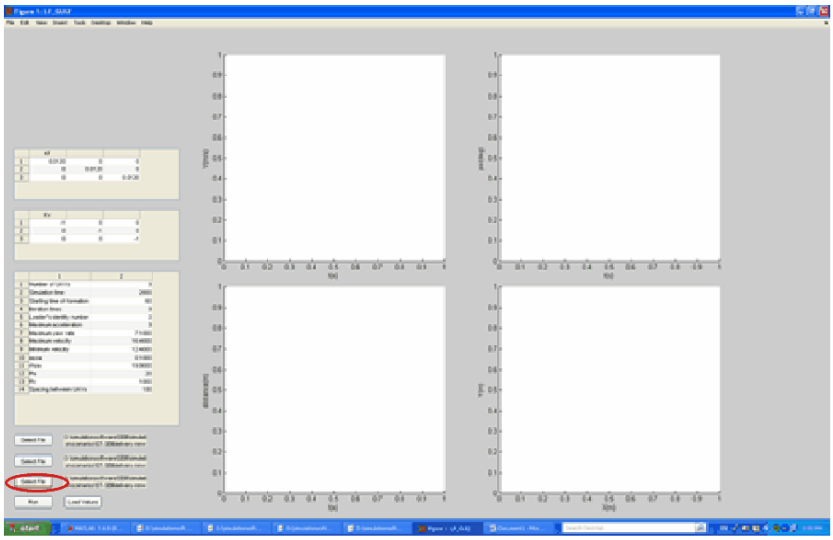


3. Click on the second “Select File” button to choose the file containing the specified

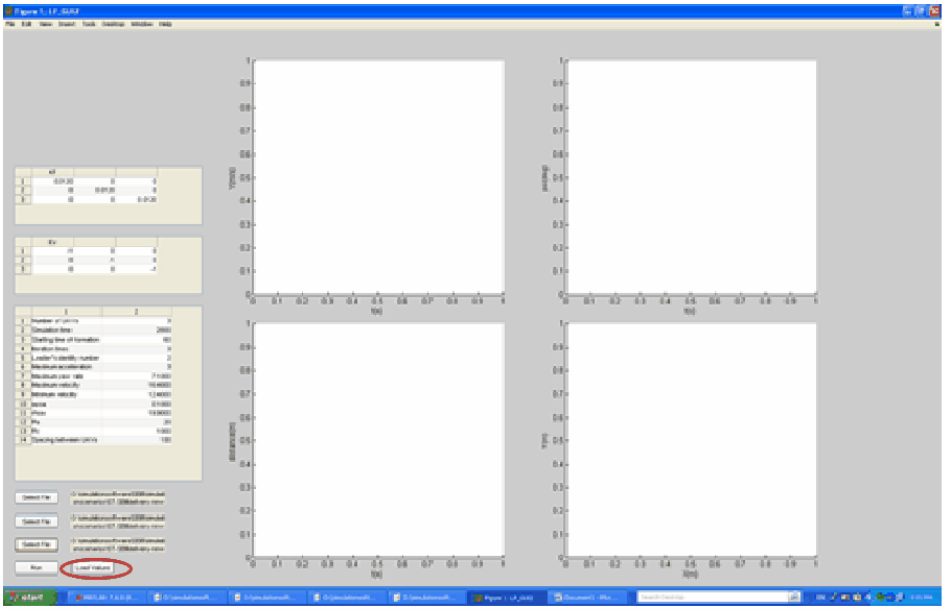
parameters in current directory. The default file name is “parametersetting.m”.



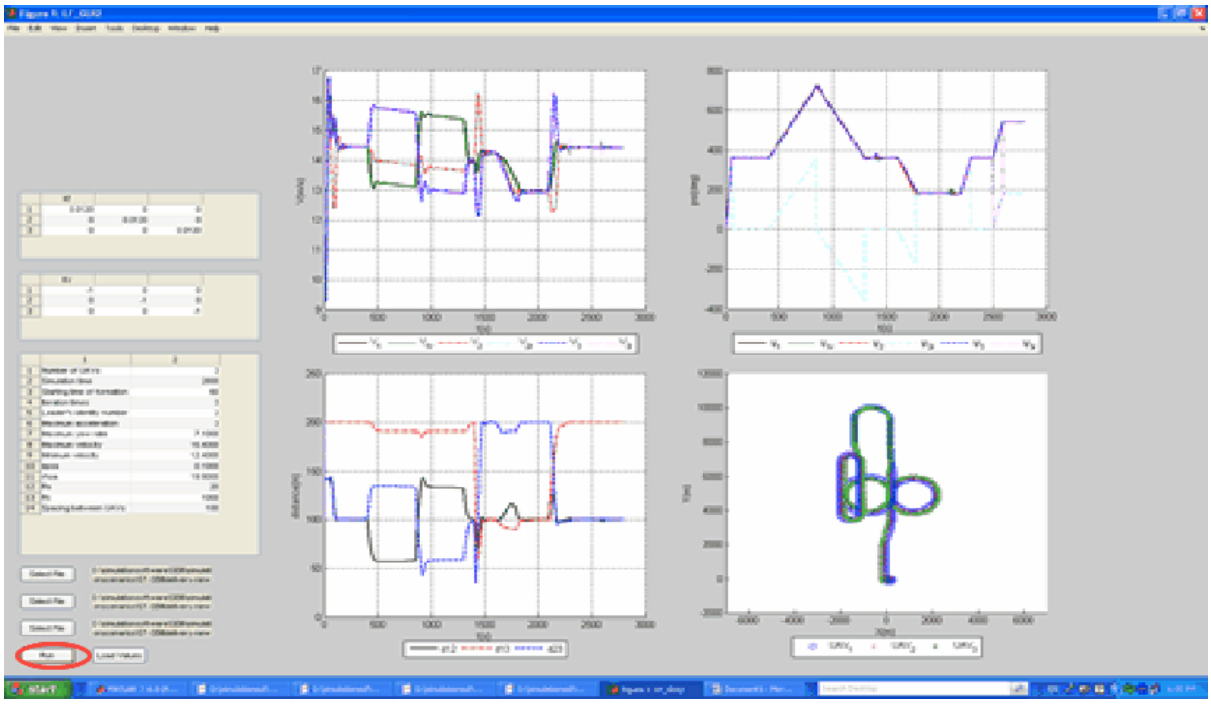
4. Click on the third “Select File” button to choose the file containing the initial states and commands of the UAVs in current directory. The default file name is “UAVinit.m”. Then you can see Tables Kf and Kv as well as the parameter table are all filled up.



5. Click on “Load Values” button to load the parameters in the tables and in the Matlab files to workspace.



6. Click on “Run” button to run the simulation. Then simulation results will be shown in the interface.



1.5 How to Set Parameters

There are three files used to set parameters: UAVinit.m, parametersetting.m and flightpath.m.

In the Matlab file “UAVinit.m”, the following initial parameters are set:

1) Total number of UAVs

```
glob.NumUAV=3;
```

2) Simulation time

```
glob.T=2800;      % s
```

3) Start time of formation control

```
glob.Ts=60;       % s
```

4) Iteration times

```
glob.loopnum=3;
```

5) Leader's identity number

```
glob.LeaderID=2;
```

6) Dynamics constraints

```
glob.amax=3;      % m/s2, maximum acceleration
glob.dpsimax=7.1; % degree, maximum yaw rate
glob.vmax=16.4;   % m/s, maximum velocity
glob.vmin=12.4;   % m/s, minimum velocity
```

7) Spacing between UAVs

```
glob.spacing=100; % m
```

8) Control parameters

```
% inner-loop control gain
Kv=-1*   [   1      0      0
             0      1      0
             0      0      1];
% outer-loop control gain
Kf=0.012*[   1      0      0
             0      1      0
             0      0      1];
% small scalar used in collision avoidance
epsa=0.1;
% parameter used in collision avoidance
rhoa=19.9;
% collision detection distance
Ra=20;
% maximum communication distance
Rc=1000;
```

9) Initial state of UAV i

```
state(1,i).x=100;          % m, East
state(1,i).y=-100;         % m, North
state(1,i).h=glob.H0;      % m, height
state(1,i).v=14.4;         % m/s, airspeed
state(1,i).gamma=0;        % rad, flight path angle
state(1,i).psi=0/glob.r2d; % rad, yaw angle
state(1,i).phi=0;          % rad, bank angle
state(1,i).omega=0;        % rad/s, roll rate
state(1,i).Cl=0;           % lift coefficient rate
state(1,i).T=0;            % thrust rate
```

10) Initial command of UAV i

```
cmd(1,i).airspd=14.4;      % m/s, airspeed command
cmd(1,i).alt=300;          % m, altitude command
cmd(1,i).yaw=0;            % degree, yaw angle command
```

11) Initial disturbance

```
disturbance = [0   % m/s, wind velocity in X direction
               0   % m/s, wind velocity in Y direction
               0   % m/s2, wind acceleration in X direction
               0]; % m/s2, wind acceleration in y direction
```

In the Matlab file “parametersetting.m”, the formation shape and disturbance are specified.

12) Formation shape

For example, in the case three UAVs fly in alongside formation with their positions in the order of UAV1, UAV2 and UAV3 from East to West, we specify that

```
h1=TT*[glob.spacing;0;0];
h2=TT*[0;0;0];
h3=TT*[-glob.spacing;0;0];
```

where TT is a transfer matrix for rotation of formation shape and “glob.spacing” is the spacing between UAVs.
In the case the three UAVs fly in tandem formation with their positions in the order of UAV2, UAV1 and UAV3 from North to South, we set

```
h1=TT*[0;0;0];
h2=TT*[0;glob.spacing;0];
h3=TT*[0;-glob.spacing;0];
```

If the three UAVs first fly in alongside formation during 0<t<Tf1, then reconfigure to tandem formation during Tf1<t<Tf2, and finally reconfigure back to alongside formation when t>Tf2, the desired formation shape can be programmed as follows.

```
if glob.tim<Tf1
      h1=TT*[glob.spacing;0;0];
      h2=TT*[0;0;0];
      h3=TT*[-glob.spacing;0;0];
      glob.hh=[h1;h2;h3];
else
      if glob.tim<Tf2
            h1=TT*[0;0;0];
            h2=TT*[0;glob.spacing;0];
            h3=TT*[0;-glob.spacing;0];
            glob.hh=[h1;h2;h3];
      else
            h1=TT*[glob.spacing;0;0];
            h2=TT*[0;0;0];
            h3=TT*[-glob.spacing;0;0];
            glob.hh=[h1;h2;h3];
      end
end
```

13) Disturbance

If there is 3m/s wind blowing Eastwards during Tdls<t<Tdle, the wind disturbance can be programmed as follows

```
if glob.tim>Td1s
      if glob.tim<Td1e
            disturbance=[3;  % wind velocity Eastwards
                         0;  % wind velocity Northwards
                         0;  % wind acceleration Eastwards
                         0]; % wind acceleration Northwards
         else
            Disturbance=[0;
                         0;
                         0;

                         0];
      end
end
```

In the Matlab file "flightpath.m", the leader's yaw angle is specified.

14) Leader's yaw angle

For example, If the leader is required to fly first northwards, then to make a clock-wise turn in 0.8 degree/s during 400<t<850s, and finally to continue to fly northwards, the leader's yaw command is programmed as follows.

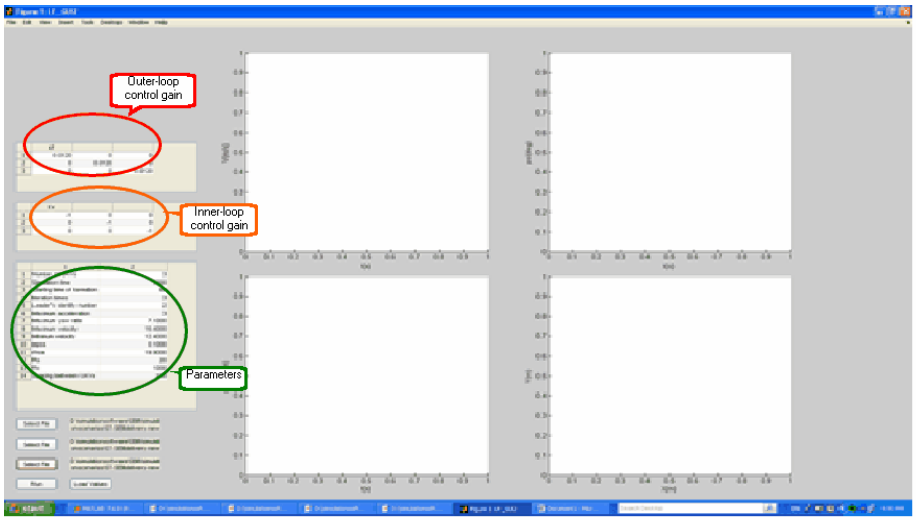
```
Tp1=400;
Tp2=850;
glob.yawrate1=0.8;

if glob.tim>Tp1
    if glob.tim<Tp2
          cmd(i,glob.LeaderID).yaw=glob.yawrate1*(glob.tim-Tp1);
    else
          cmd(i,glob.LeaderID).yaw=0;

    end
else
    cmd(i,glob.LeaderID).yaw=0;
end
```

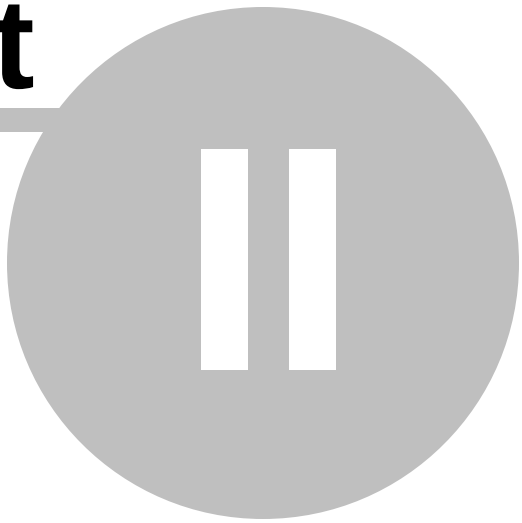
1.6 How to Modify Parameters

Many of the parameters can be modified by keying in the new values in the tables on the mini-toolbox interface.



However, the desired formation shape, the imposed wind disturbance and the specified leader's yaw command and the initial states and commands of the UAVs need be modified in the corresponding Matlab file.

Part



2 Target detection, image tracking and data fusion

2.1 Target Detection Toolbox

A software toolbox, called “VisDet”, has been developed to test the proposed target detection and image tracking algorithms using C++ language. This toolbox has the following features.

1. The proposed vision algorithms, including target detection, image tracking and data fusion, have been packaged in a Class using C++ language. The algorithms have also been tested in an onboard computer of the UAV in real flight.
2. Provide a user interface to visualize the processing results and evaluate the detection performance in real time (see Fig. 5.1 and 5.2).

Computer Requirements

The following are minimum requirements for the installation of the “VisDet” Software:

1. CPU: Pentium-class (1.3GHz minimum)
2. RAM Memory: 1GB minimum, 2GB+ recommended
3. Hard Drive Free Memory: 512MB
4. Operating System: Windows XP, or Windows 7

2.2 Installation

To install Target Detection Toolbox onto your computer with Windows:

1. Locate the Vision Detection folder. Double click on the setup.exe file.
2. Follow the setup wizard instructions. You will install toolbox.

2.3 Quick Start

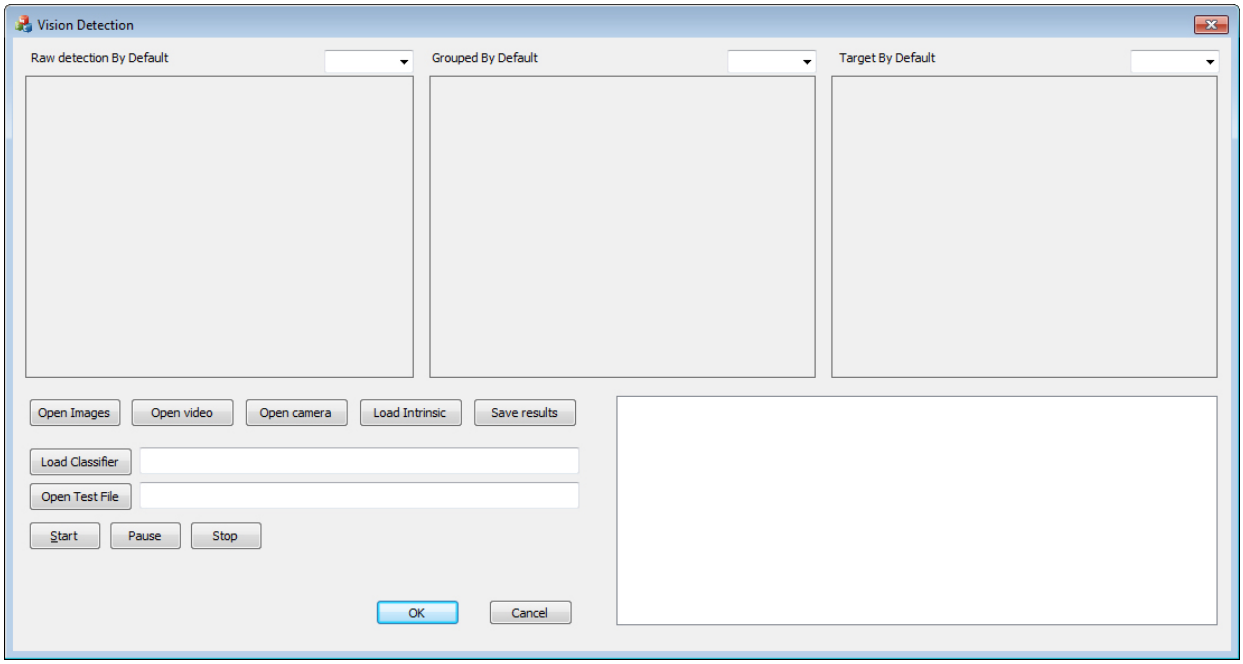


Figure 5.1: GUI of the toolbox.

The procedure to use the toolbox is give as follows:

- Step 1. Open images, video or a camera for the processing. The supported image formats includes: jpg and bmp, and the supported video format is .avi. The only supported camera is the pointgrey firefly, which is as same as the one used in the UAV. The sample images for testing can be found in “test pic” under the setup folder.
- Step 2. Open a classier. Three pre-trained classifiers have found in “classifier” under the setup folder. The description of those classifiers has been given in Table 5.1.
- Step 3. Press run. The software will autonomous detect and track the target (a quadrotor UAV) in the image. The tracking results will be shown in the status window.

Table 5.1: Classifiers

Classifiers	Classifier Description
Classifier_trainout_quadrotor_outdoor_PGR_3000 11000.xml	Quadrotor detection in outdoor environment trained using 3000 positive and 11000 negative

	images
Classifier_trainout_quadrotor_outdoor_PGR_1600_6000.xml	Quadrotor detection in outdoor environment trained using 1600 positive and 6000 negative images
Classifier_trainout_quadrotor_sky_PGR_3000_7000_12.xml	Outdoor detection in sky background trained using 3000 positive and 7000 negative images

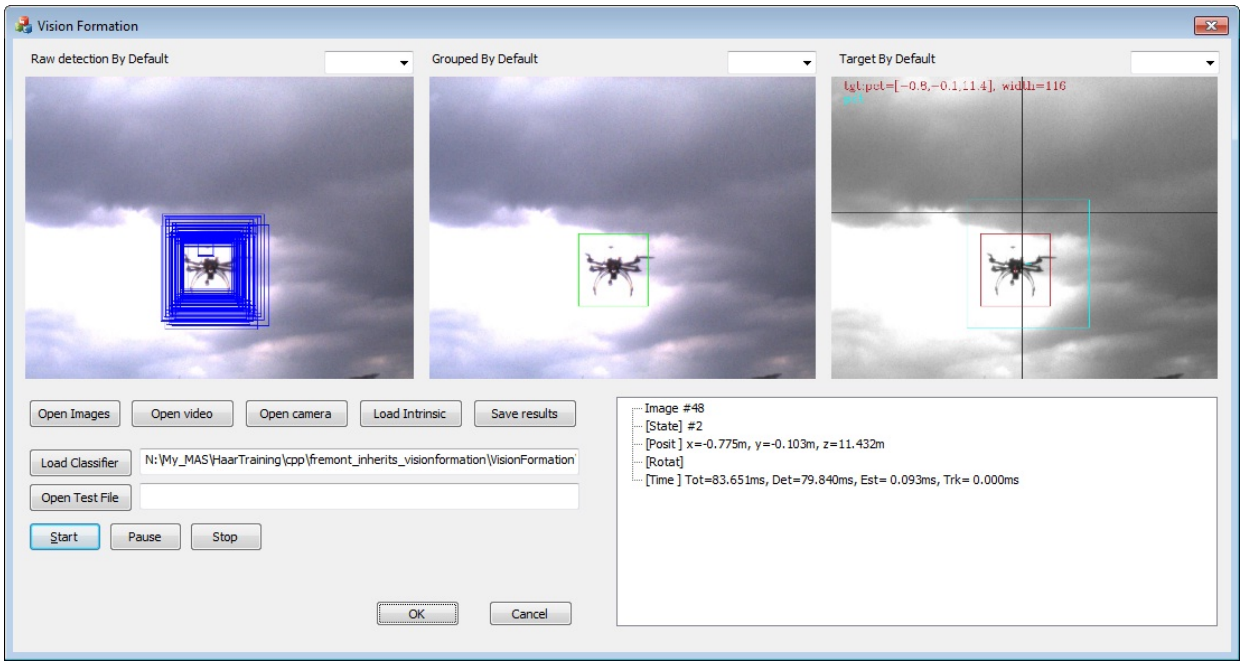


Figure 5.2: GUI of the toolbox in operation.

The parameters for the vision algorithms have been defined in a configuration file: "DetUAVConfig.text". User can change the parameters, and the definition of the parameters has been given in Table 5.2.

Table 5.2: The parameters defined in "DetUAVConfig.text"

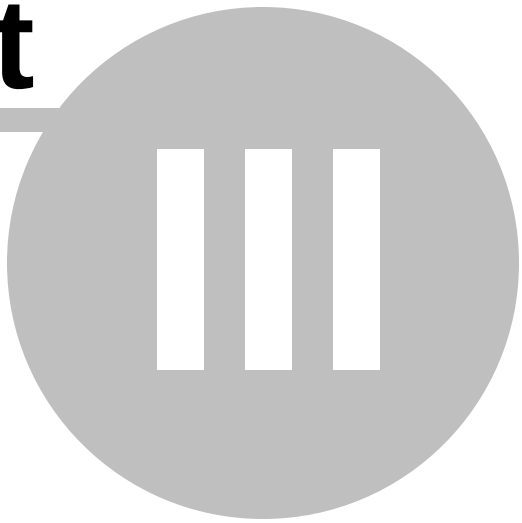
Item	Default Value	Description

bFrame_320_240	0	The size of the images. 0: 640 × 480, 1: 320 × 240
MIN_NEIGHBORS_S1	15	Min neighbours in the detection stage
MIN_NEIGHBORS_S2	4	Min neighbours in the tracking stage
scaleFactor _S1	1.05	Scale factor in the detection stage
scaleFactor_S2	1.05	Scale factor in the tracking stage
minSizeHaarWindow	16	Minimal size of the Haar window
minSizeHaarWindow	220	Maximum size of the Haar window
M_TrackSigma	10	Covariance of the Kalman filter

Notice

This software is a beta version. If users find any problems with this software, please feel free to contact: tsllinf@nus.edu.sg.

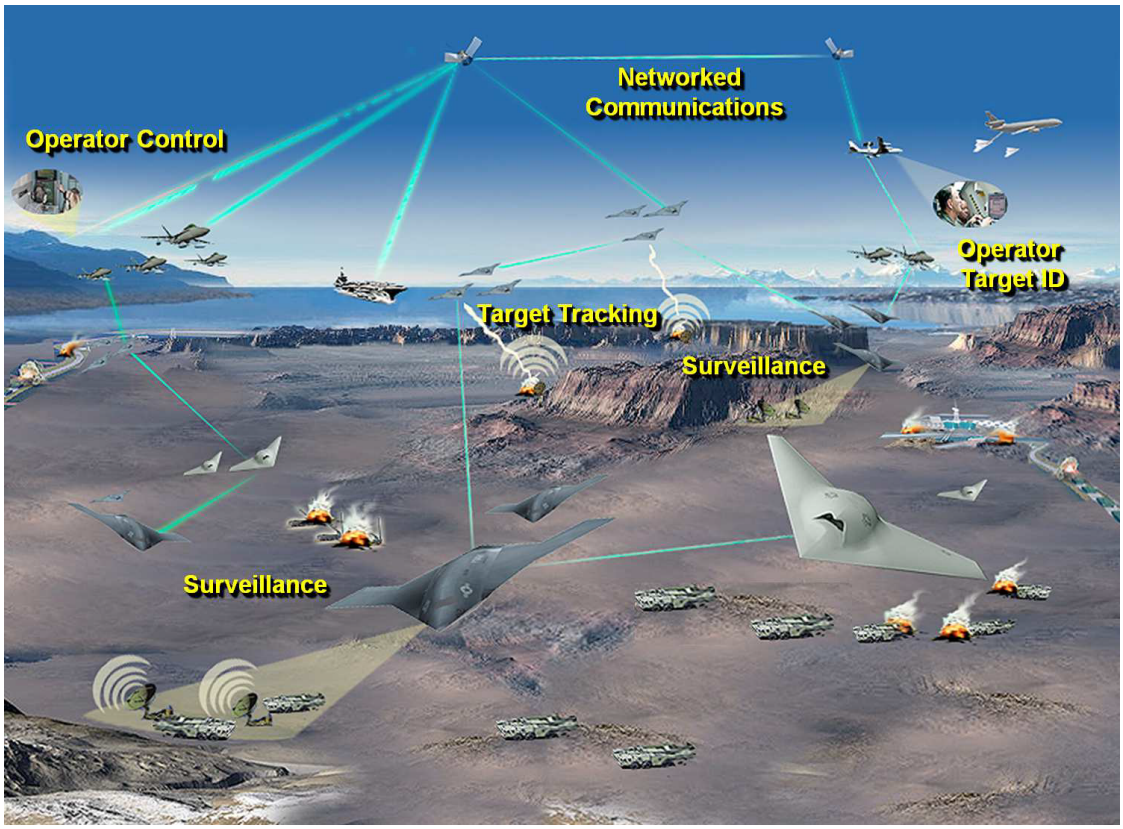
Part



3 Distributed Tasking

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.

3.1 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

3.1.1 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 • Variations to request with minimal change if

Item	Description
	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.

3.1.2 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

3.1.3 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*

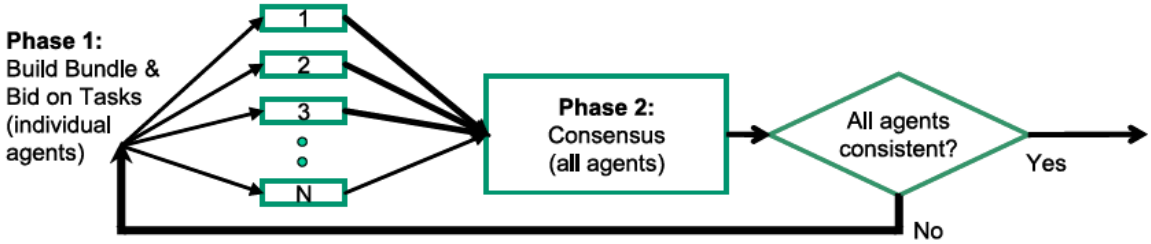
3.1.4 Algorithm Procedure

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.***



3.1.4.1 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.

3.1.4.2 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.*

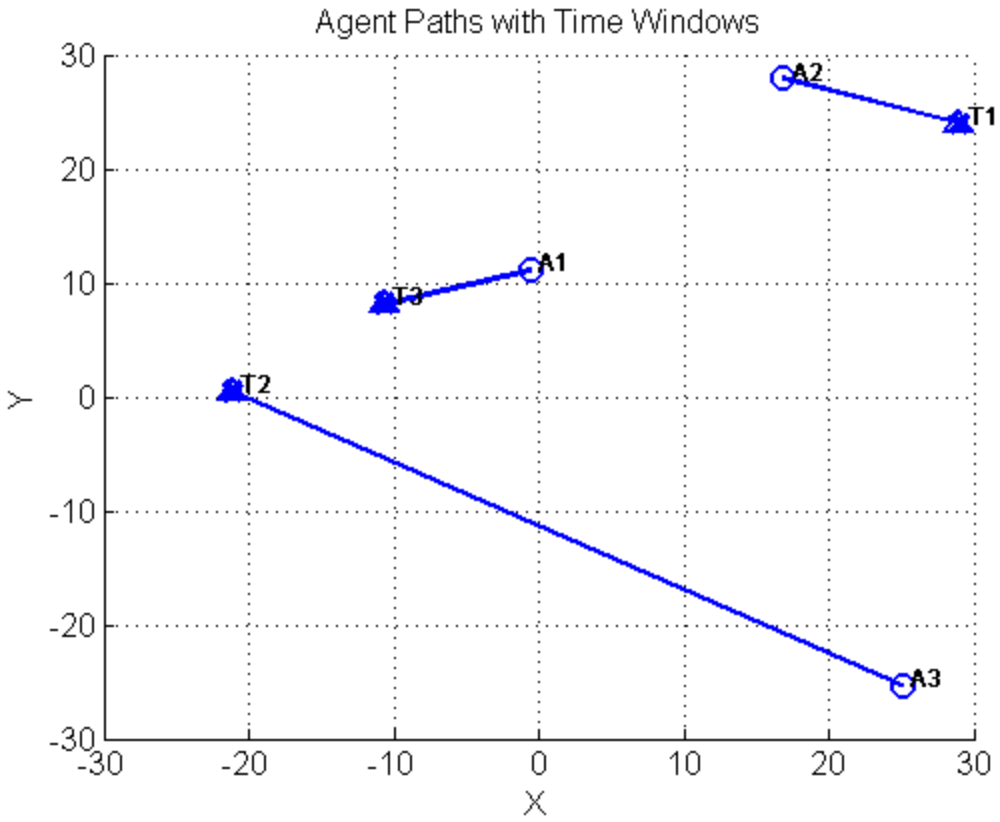
And then using some communication protocols and "design rules" the algorithm will assign the best UAV to perform the corresponding task. From here, the algorithm returns to the first phase where new tasks can be added to the bundle. CBBA will iterates between these two phases until no changes to the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

3.1.5 Algorithm Properties

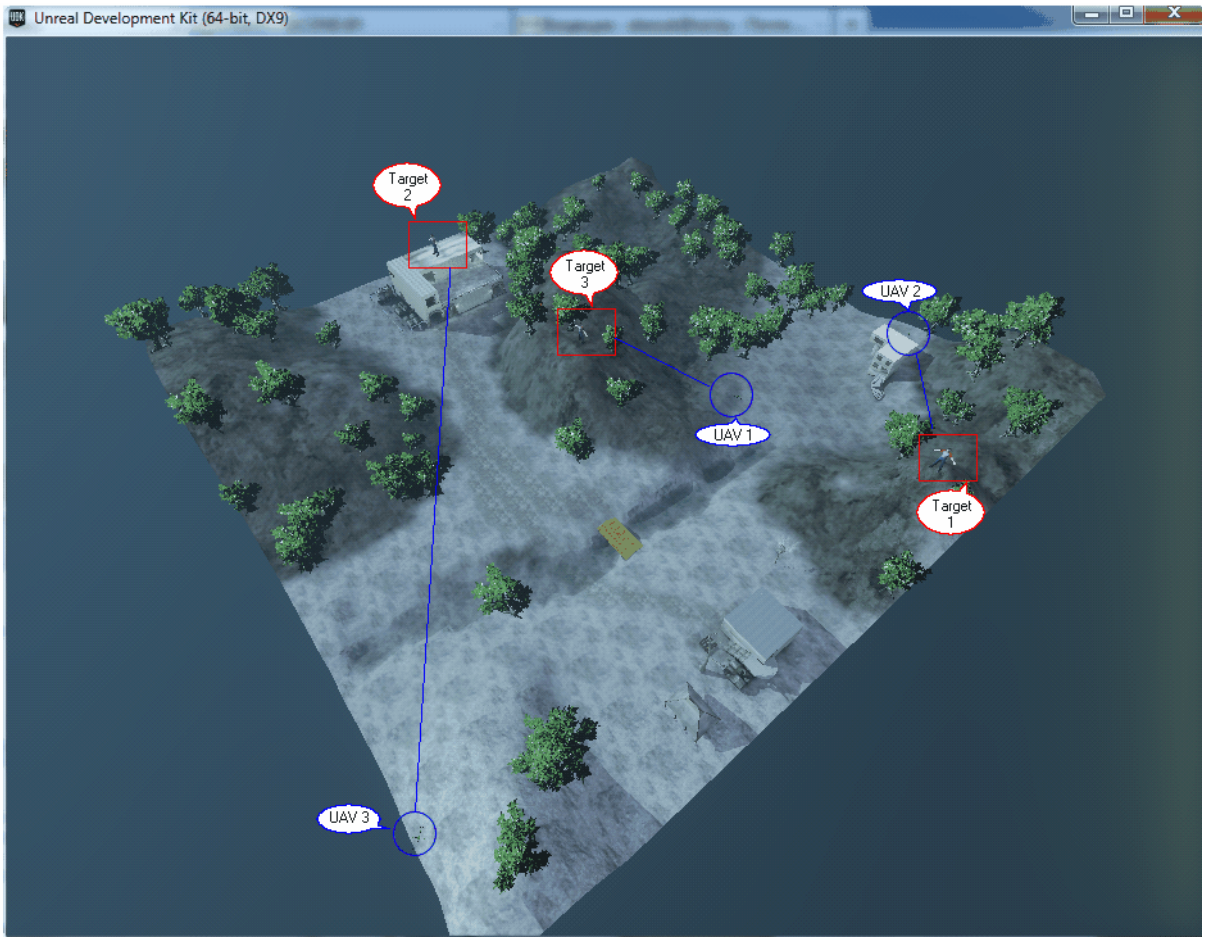
- Task selection - Polynomial-time, provably good feasible solutions
- Guaranteed real-time convergence even with inconsistent environment knowledge
- Time-varying score functions (e.g. time-windows of validity for tasks)

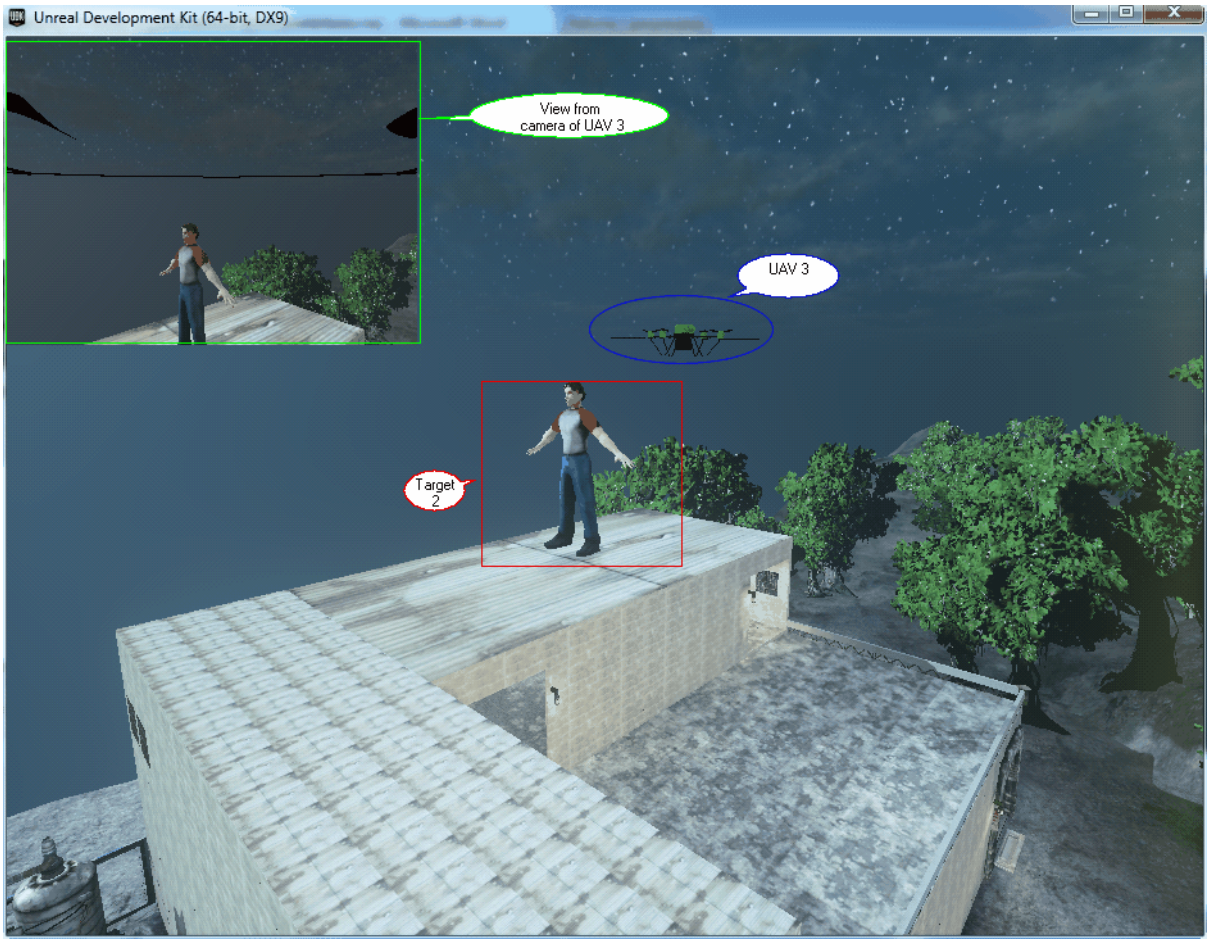
3.1.6 Example

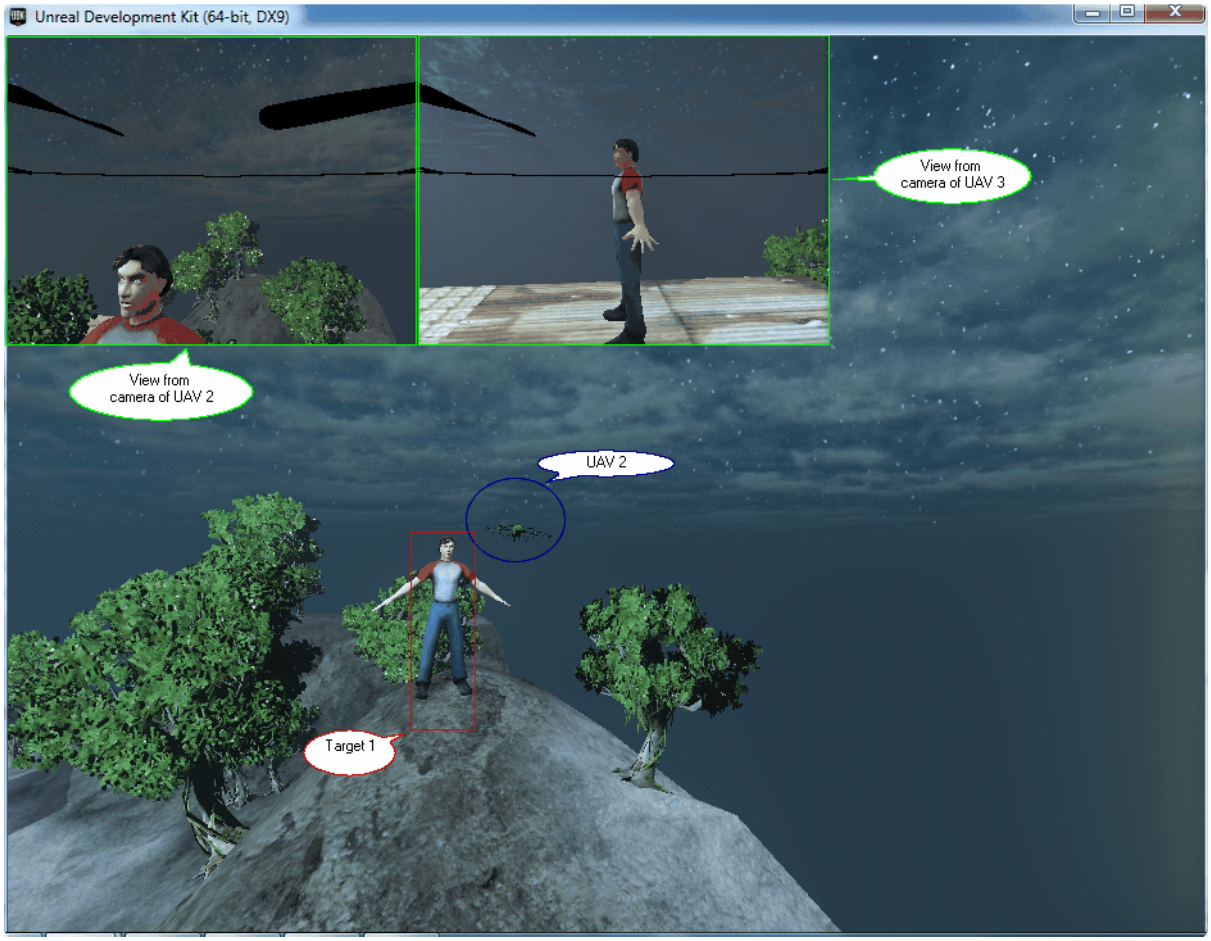
For example the so called Tethered UAVs Self-Assignment Problem is a particular case of the problem described above. And the solution of this particular problem presented in a figure below, namely, we find the logic that enabled the tethered UAVs (3 UAVs) to self-deploy one UAV to each specified location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



Start position:





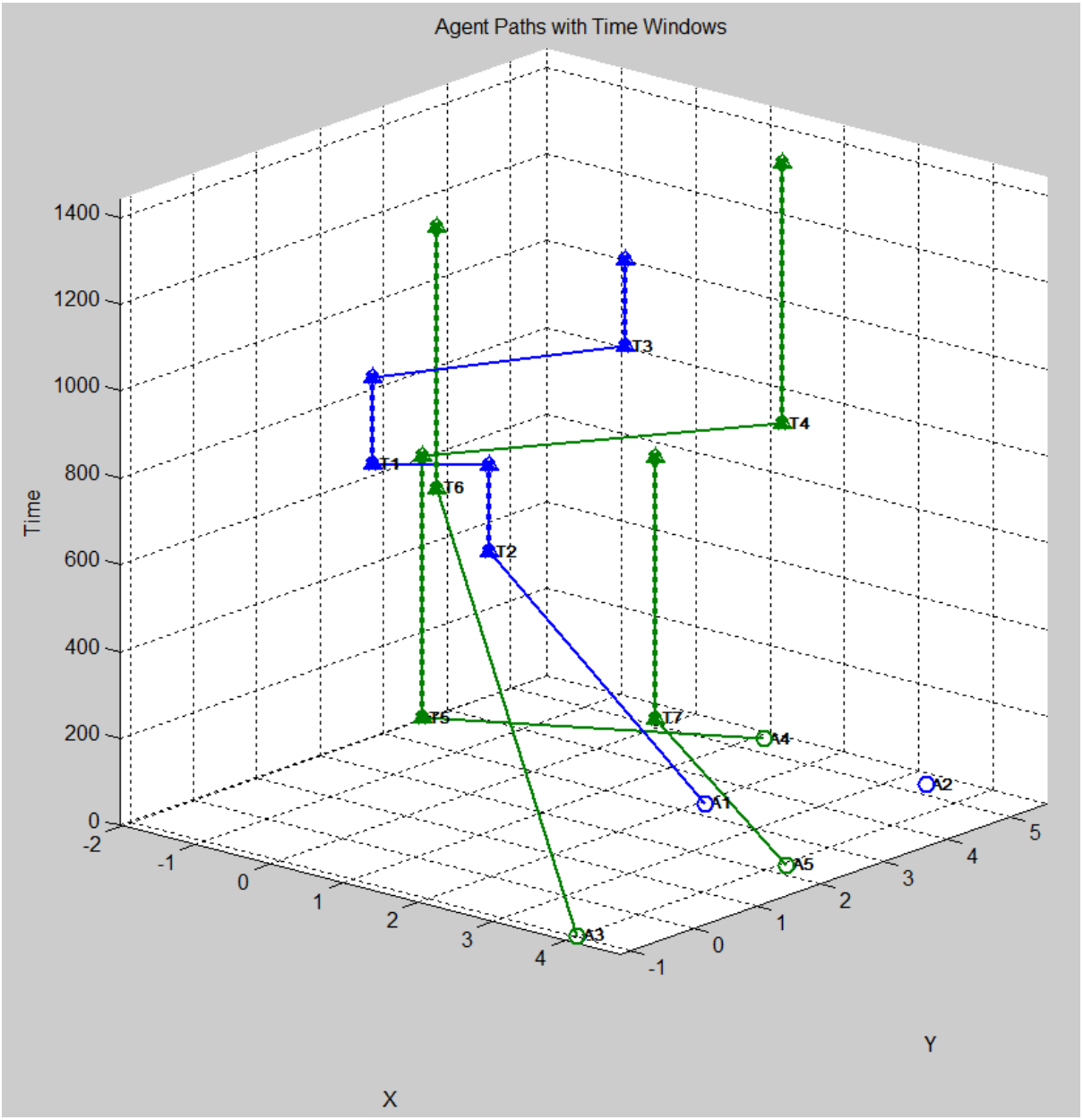


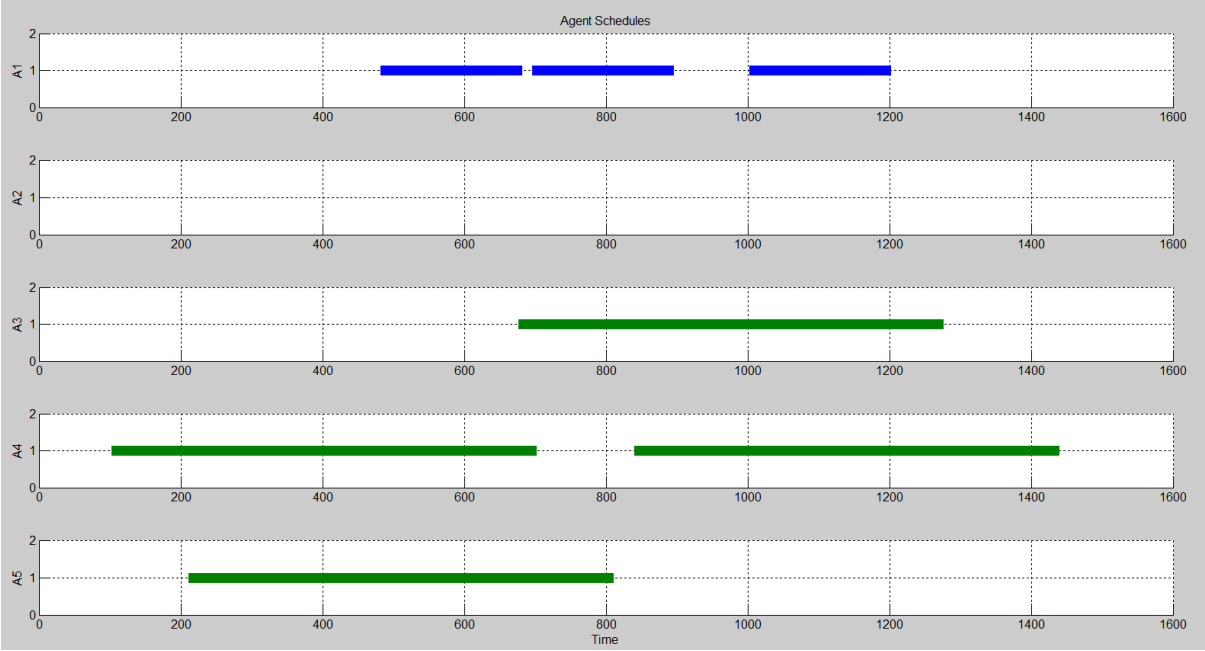
3.1.7 Example 2

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).





3.1.8 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.)

3.2 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.

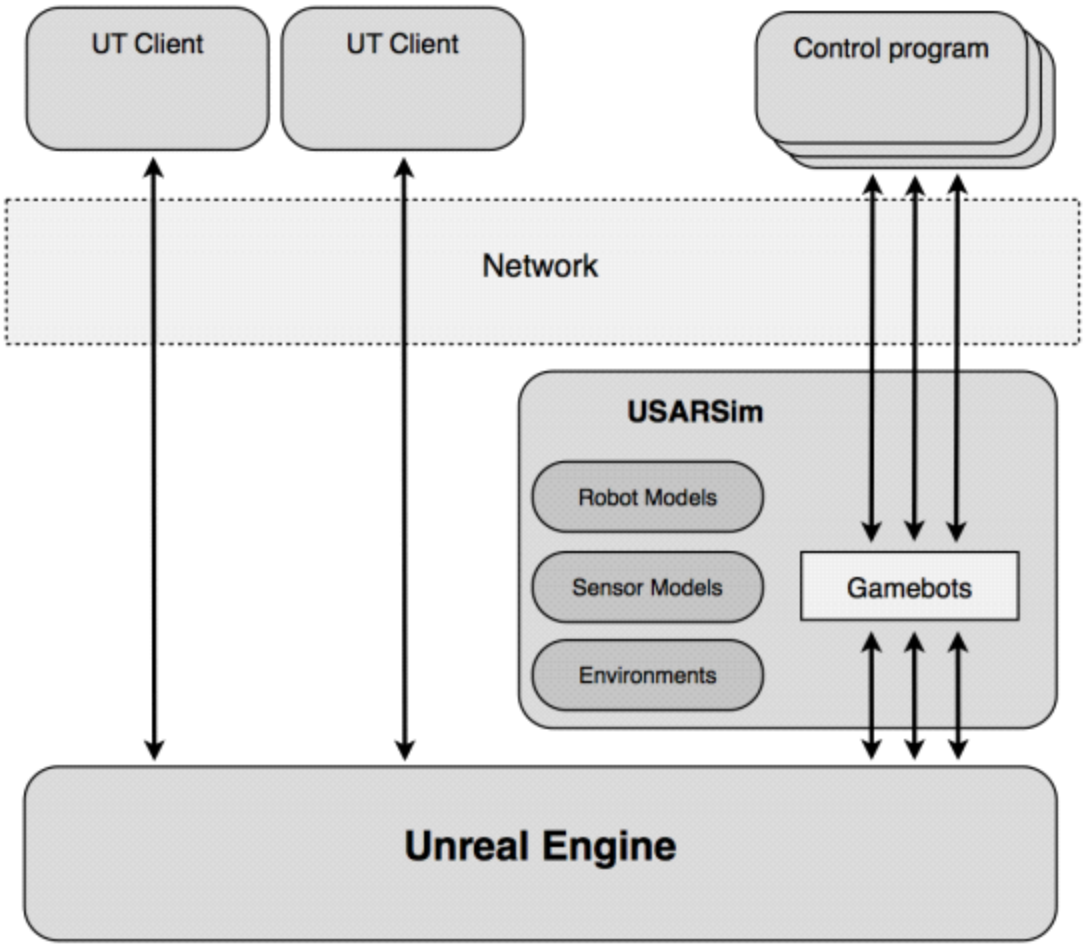


Figure 1.1: USARSim architecture

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

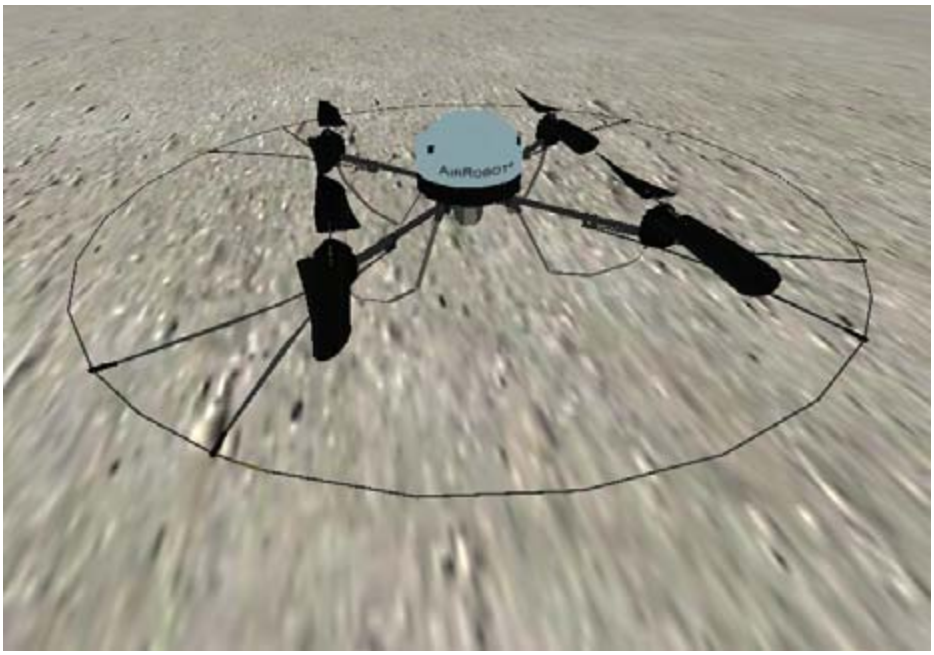


Figure 1.3: Simulated AirRobot

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.

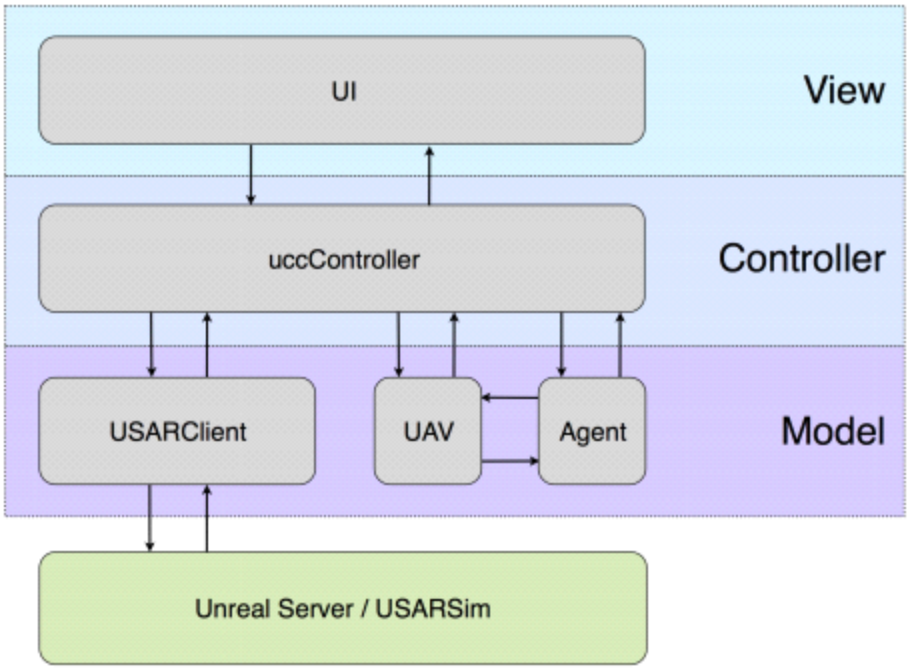


Figure 1.3: Control program architecture overview

3.2.1 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.

3.2.2 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.

3.2.3 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.

3.2.4 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.

3.2.5 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.*

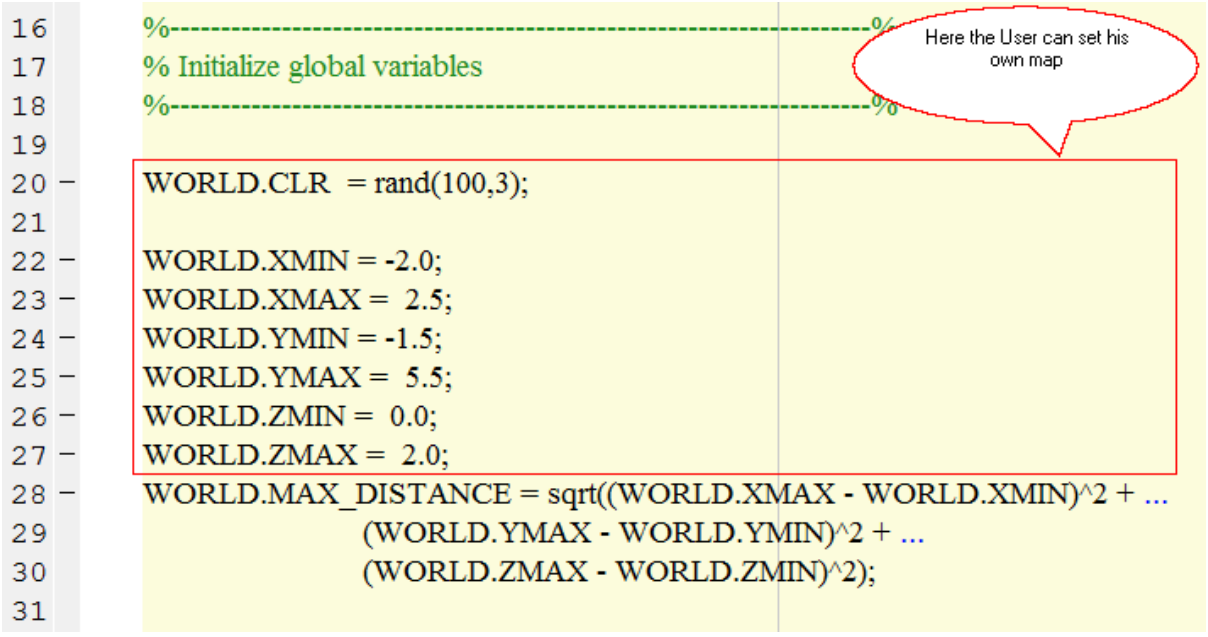
9) PlotAssignmennts.m

Description: *Plot CBBA output*

3.2.6 How to Change the variables

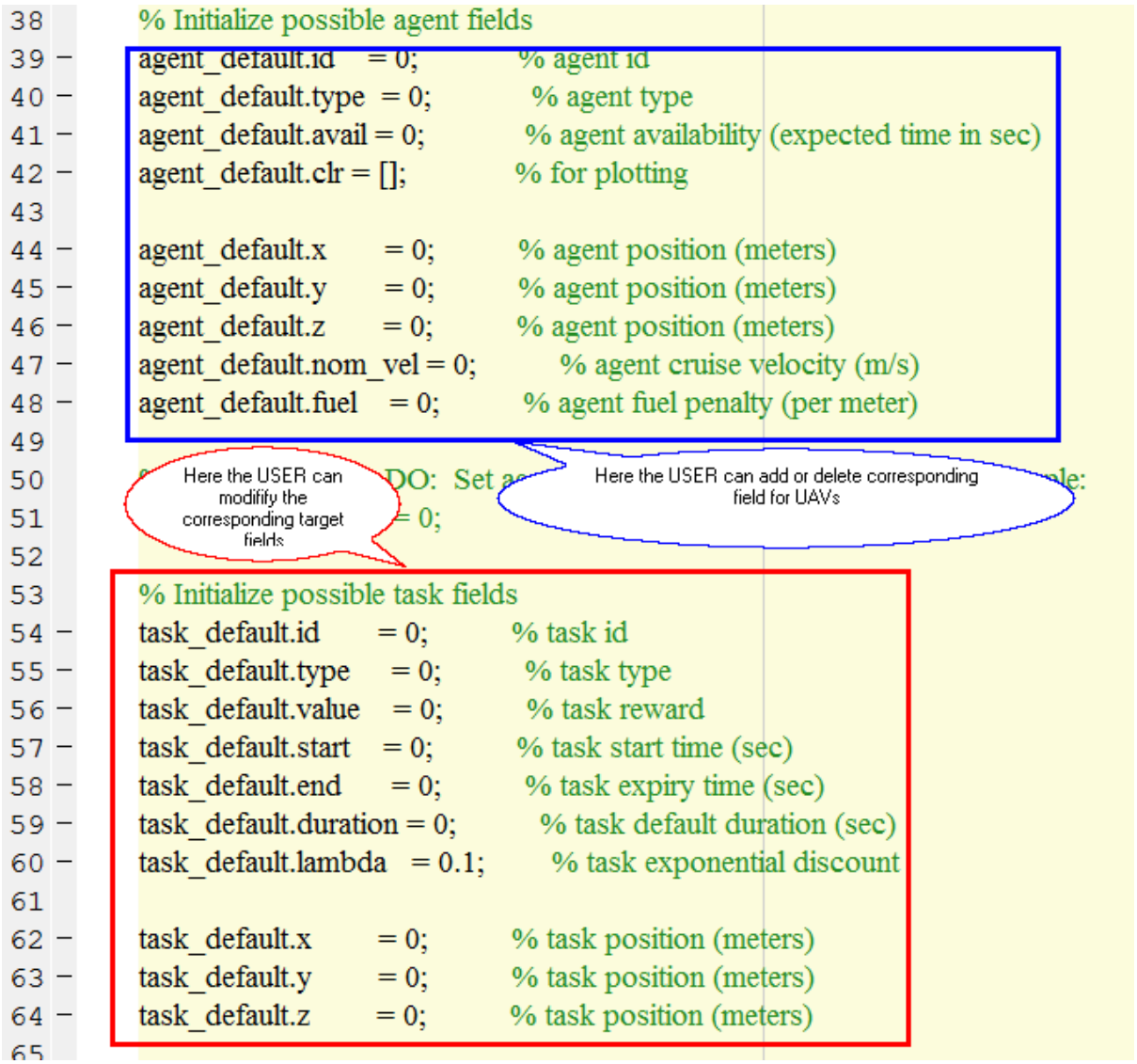
The USER should open the file MainTestScript.m
and he can modify the following parameters:

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
```

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)

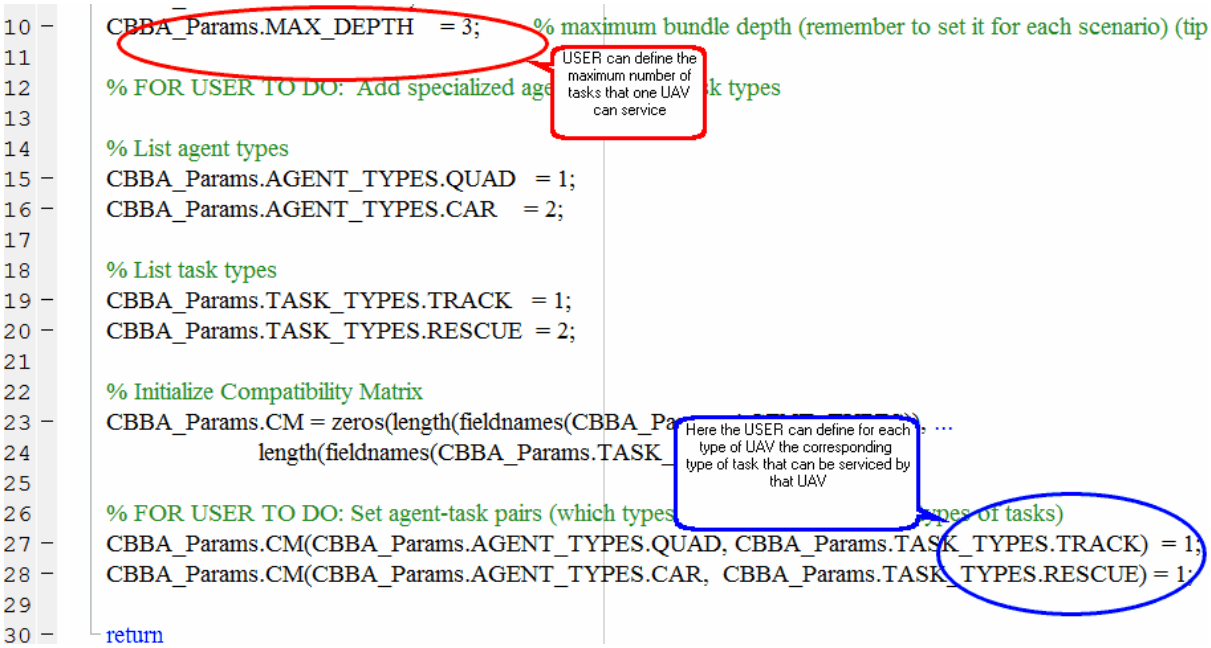
3. Select the total number of UAVs and Tasks which will be simulated in the scenario

```
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)



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

Part

4 Distributed Target Search Algorithm

4.1 Problem Class Description

1) Cooperative coverage and search of multiple ground targets by a group of UAVs with limited sensing and communication capabilities.
2) The algorithms are fully autonomous and distributed.
3) The proposed methods are vision-based and can be applied to a 3D dynamic environment.
4) The surveillance regions can be convex or non-convex.

4.2 Solution Description

Here, we compare the benefits of the proposed search algorithm with conventional methods in the literature

Benefits Compared with Existing Methods

Algorithms	2D	3D	Environment Map		Information fusion	Moving Target	Region	
			Static	Dynamic			Convex	Non-Convex
Our method	✓	✓	✓	✓	✓	✓	✓	✓
L. Pimenta *et al*[1]	✓		✓	✓		✓	✓	
M. Schwager *et al* [2]	✓	✓	✓				✓	✓
M. Schwager *et al* [3]	✓		✓	✓	✓		✓	
P. Millet *et al* [5]	✓		✓	✓	✓		✓	

I. I. Hussein *et al* [6]	✓		✓				✓	✓
A. K. Sun *et al* [7]	✓		✓	✓	✓	✓	✓	
J. Tisdale *et al* [8]	✓		✓	✓			✓	
J. Berger *et al* [9]	✓		✓	✓	✓		✓	
M. Zhong *et al* [10]	✓		✓	✓			✓	
J. Cortes *et al [11]*	✓		✓				✓	

The main features and advantages of the proposed fully autonomous and distributed coverage and search algorithm are

- Collaborative fusion and control of UAVs with limited sensing and communication capabilities in unknown environments
- Vision-based time-varying detection probability model of UAVs
- Distributed data fusion scheme and control strategy using only local measurement and neighbors' observations.
- Capability in both convex and non-convex regions and capability for both heterogeneous and homogeneous systems

Limitations:

The proposed method is more general than the existing methods, and has wider applications. Due to the information fusion step involved and complicated objective function, the computational complexity and communication load are slightly higher than other methods.

4.3 Brief Description of Solution

Algorithm Input/Output and Parameters to be set

Note: The target search algorithm is to be on each UAV, running at every time step, e.g., 0.1 second.

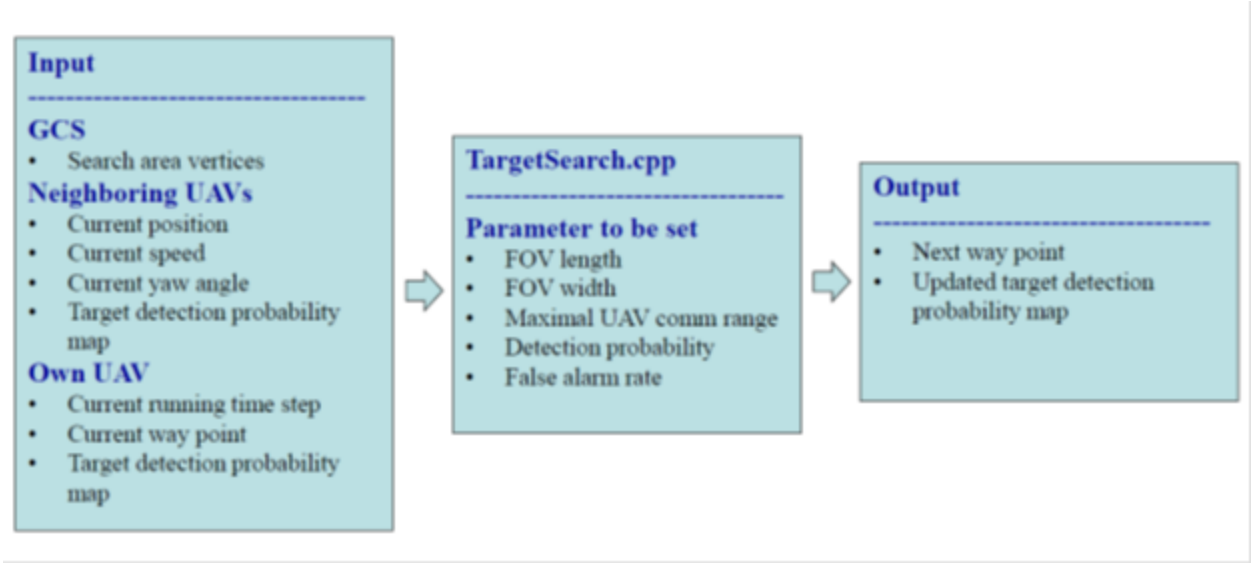


Fig. 1 Input/Output of algorithm and parameter to be set

Note: The algorithm requires that the following for each UAV to broadcast to its neighboring UAVs at every time steps

1) UAV positions
2) UAV speed
3) Heading angle (yaw angle)
4) Target detection probability map

The information from neighbors is used for UAV to update the probability map and decide its next way point. Also the information will be used for collision avoidance.

Algorithm Procedure

Step 1: At time step k, according to the camera measurement, UAV update its own probability map

Step 2: UAV exchanges the information with its neighbors. The information includes: UAV position, UAV speed, yaw angle and probability map.

Step 3: UAV updates its probability map according to its neighbors' information

Step 4: In the Voronoi region assigned to UAV at time step k, UAV calculates the centroid of the probability map or finds the cell which associates with the highest target existence uncertainty.

Step 5: UAV decides its next way point according to the position of centroid or the maximal uncertain cell. UAV turning rate constraint will also be considered at this step.

Step 6: Collision avoidance. UAV decides to avoid the collision with its neighbors if their positions are within a predefined safe region.

Algorithm Properties

If the target detection probability and false alarm rate is larger than 0 and less than 1, then the algorithm can guarantee that all the targets in the search area can be found, if the search time is long enough.

4.3.1 Example

1) Search region:

Square area, 2 km * 2 km

2) Constraints of the UAV:

Parameter	Max / Min Values
Flight path angle	13.0 deg (up) / 0 deg (TBC)
Bank angle	12 deg / -12 deg
Air speed	16.4 m/s / 12.4 m/s
Acceleration (forward direction)	3.0 m/s^2 / 0 m/s^2 (TBC)
Turn rate	7.1 deg/s / 0 deg/s

3) Parameters of onboard camera

Square FOV: length 150 m, width 150 m

4) Other parameters

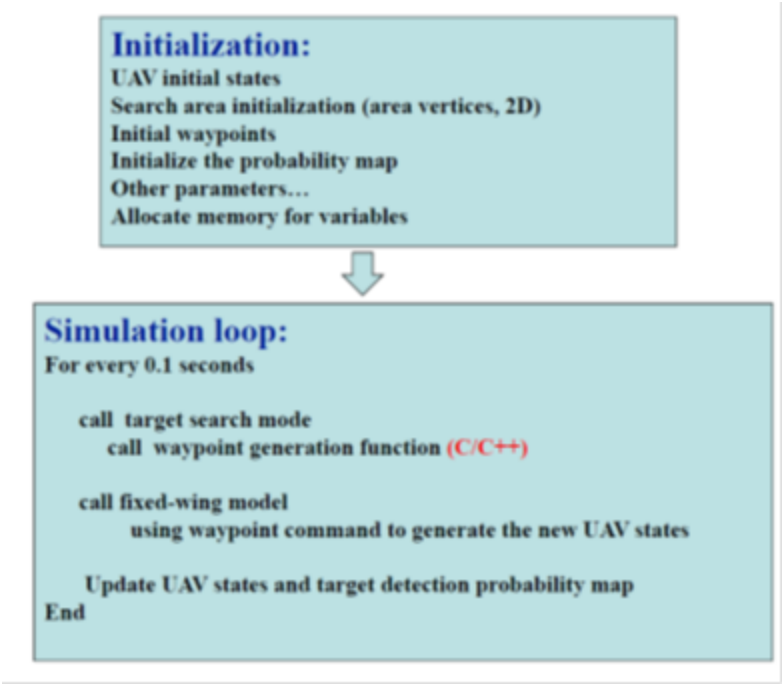
The number of UAVs: 3

Maximal communication range: 3000 m

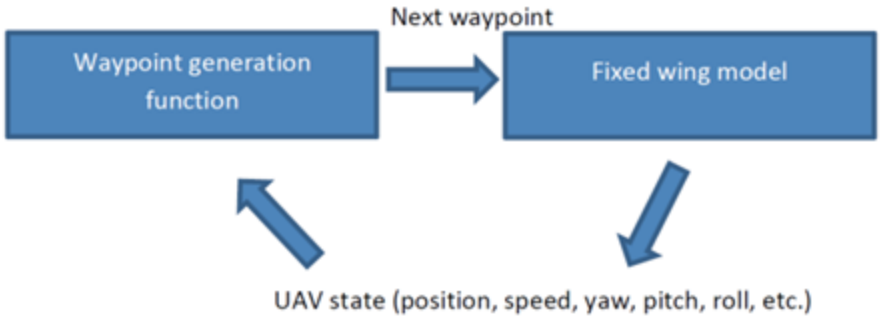
Target detection probability: 0.9

Target detection false alarm: 0.1

5) Proposed algorithm structure: (how to use the search function in our case)



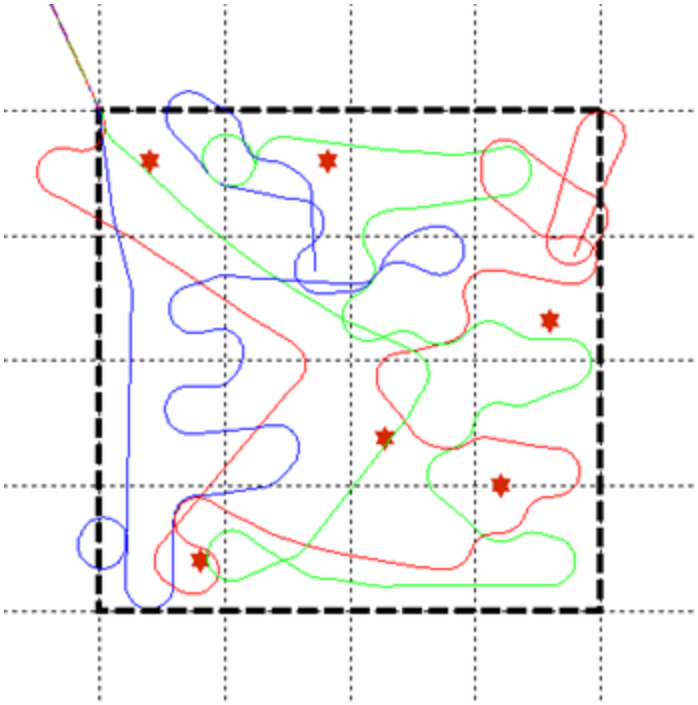
(a) Whole structure of search algorithm



(a) Waypoint generation part

Fig. 2 Current search algorithm function

1) Simulation results



4.4 Mini-toolbox and development / test environment

Matlab and C toolboxes for the algorithms are attached.

4.5 Open research questions

The following problems remain open at the end of the project and can be considered for future work:

- More efficient communication protocol during information fusion processing. The current algorithm involves the probability map transmission among UAVs. The communication burden is high especially when the search area is large.
- The multiple-objective search problems, e.g., time-efficient and full-coverage requirements. Currently, search time has not been considered in the objective function even though the algorithm does try to minimize the overall search time (**around 10 mins using 6 UAVs, 2k*2k area, 15 mins using 3 UAVs, 2k*2k area**). In future, we want to consider the multiple-objective search problem to minimize the search time and guarantee the full coverage of search area.
- Optimal search under 3D cluttered urban environment due to line-of-sight problem.

References:

[1] Luciano C. A. Pimenta and Mac Schwager, et.al, "Simultaneous Coverage and Tracking (SCAT) of Moving Targets with Robot Networks", WAFR'08, pp. 85-99.
[2] M. Schwager, B. Julian, M. Angermann and D. Rus, “Eyes in the Sky: Decentralized Control for the Deployment of Robotic Camera Networks,” Proceedings of the IEEE, September, 2011, vol. 99, no. 9, pp. 1541-1561.
[3] M. Schwager, D. Rus and J. J. Slotine, “Decentralized, Adaptive Coverage Control for Networked Robots,” International Journal of Robotics Research, March, 2009, vol. 28, no. 3, pp. 357-375.
[4] M. Schwager, D. Rus and J. J. Slotine, “Unifying Geometric, Probabilistic, and Potential Field Approaches to Multi-Robot Deployment,” International Journal of Robotics Research, March, 2011, vol. 30, no. 3, pp. 371-383.
[5] P. Travis Millet, David W. Casbeer,‘’Multi-agent Decentralized Search of a Probability Map with Communication Constraints,” AIAA Guidance, Navigation and Control Conference, pp. 1-12, 2010.
[6] Islam I. Hussein and Dusan M. Stipanovic, ‘’Effective Coverage Control for Mobile Sensor Networks with Guaranteed Collision Avoidance,’’ IEEE Tran. Control Systems Technology, vol. 15, no. 4, pp. 642-657, 2007.
[7] Andrew K. Sun and Hugh H. T. Liu, “Cooperative UAV Search for Moving Targets Using a Modified Diffusion Uncertainty Model,” AIAA Guidance, Navigation and Control Conference, pp. 1-19, 2009.
[8] John Tisdale, Z. Kim and J. Karl Hedrick, “Autonomous UAV Path Planning and Estimation,” IEEE Robotics & Automation Magazine, pp. 35-42, 2009.
[9] J. Berger and J. Happe, “Co-evolutionary search path planning under constrained information-sharing for a cooperative unmanned aerial vehicle team,” in Proceedings of IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
[10] M. Zhong and C. Cassandras, “Distributed coverage control and data collection with mobile sensor networks,” in Proceedings of the IEEE Conference on Decision and Control, 2010, pp. 5604–

5609.
[11] J. Cortes, S. Martinez, T. Karatas and F. Bullo, Coverage control for mobile sensing networks, IEEE Tran. Robotics and Automation, Vol 20, No 2, pp 243-255, 2004
[12] M. Polycarpou, Y. Yang, and K. Passino, "A cooperative search framework for distributed agents," in Proceedings of the IEEE International Symposium on Intelligent Control, 2001, pp. 1–6.

Part V

5 Distributed Air-net Coverage Algorithm

5.1 Problem Class Description

1) Cooperative air-net coverage by a group of UAVs with limited sensing and communication capabilities.
2) The algorithms are fully autonomous and distributed.

5.2 Solution Description

Here, we compare the benefits of the proposed search algorithm with conventional methods in the literature

Benefits Compared with Existing Methods

The main features and advantages of the proposed fully autonomous and distributed algorithm are

- Optimizing traveling time
- Robust to UAV failures
- Distributed decision making

Limitations:

Currently, we can only obtain the suboptimal solution in term of time efficiency.

5.3 Brief Description of Solution

Algorithm Input/Output and Parameters to be set

Algorithm input:

1) UAV initial positions
2) UAV speed
3) Area of operation (boundary of surveillance area)
4) UAV communication range

Algorithm output:

1) UAV next waypoint

Algorithm Procedure

Step 1: Initialization section: initialize parameters required in algorithm input.

Step 2: Coverage control section: 1) Neighbors detection and communications; 2) Voronoi partition; 3) calculate area coverage control cost function; 4) traveling time consideration into cost function; 5) calculate next waypoint

Step 3: Connectivity maintenance

Algorithm Properties

If UAV can communication with its neighbors, then the algorithm can guarantee that UAVs can coverage the whole area gradually.

5.4 Example

Air-net UAVs are responsible for providing communication coverage in the UAV Airspace which is where the Tethered and Service UAVs are expected to operate in. In this way, all the UAVs and ground users are networked with each other at all times.

In this section, we test the distributed coverage control strategy by simulation. The code for the control algorithm is written in Matlab. The fixed-wing UAV model is integrated in UDK. The map we used is the same with which used in the last Section. Six fixed-wing air-net UAVs are positioned to cover a squared region [0, 1000] m^2. The communication range of each UAV is RC = 380m. The speed of UAVs is 10 m/s. The turning rate is 6 degree/s.

Figure 1 shows the movements of the UAVs covering the monitoring region, where the black circles enclose the communication regions and the UAVs are denoted by black points. It is demonstrated that the UAVs can achieve the desired full air-net coverage.

The trajectory of six fixed-wing UAVs is shown in Figure 2. From the figure, we can see that in the initial step, the UAVs are placed closed to each other. Then after several steps, by using the proposed distributed coverage control algorithm, the UAVs will autonomously deploy themselves to the right positions that will provide air-to-air communications coverage of the UAV Airspace. In this simulation, the movement constraints of fixed-wing UAVs are

also considered, such as the speed, turning rate, etc.

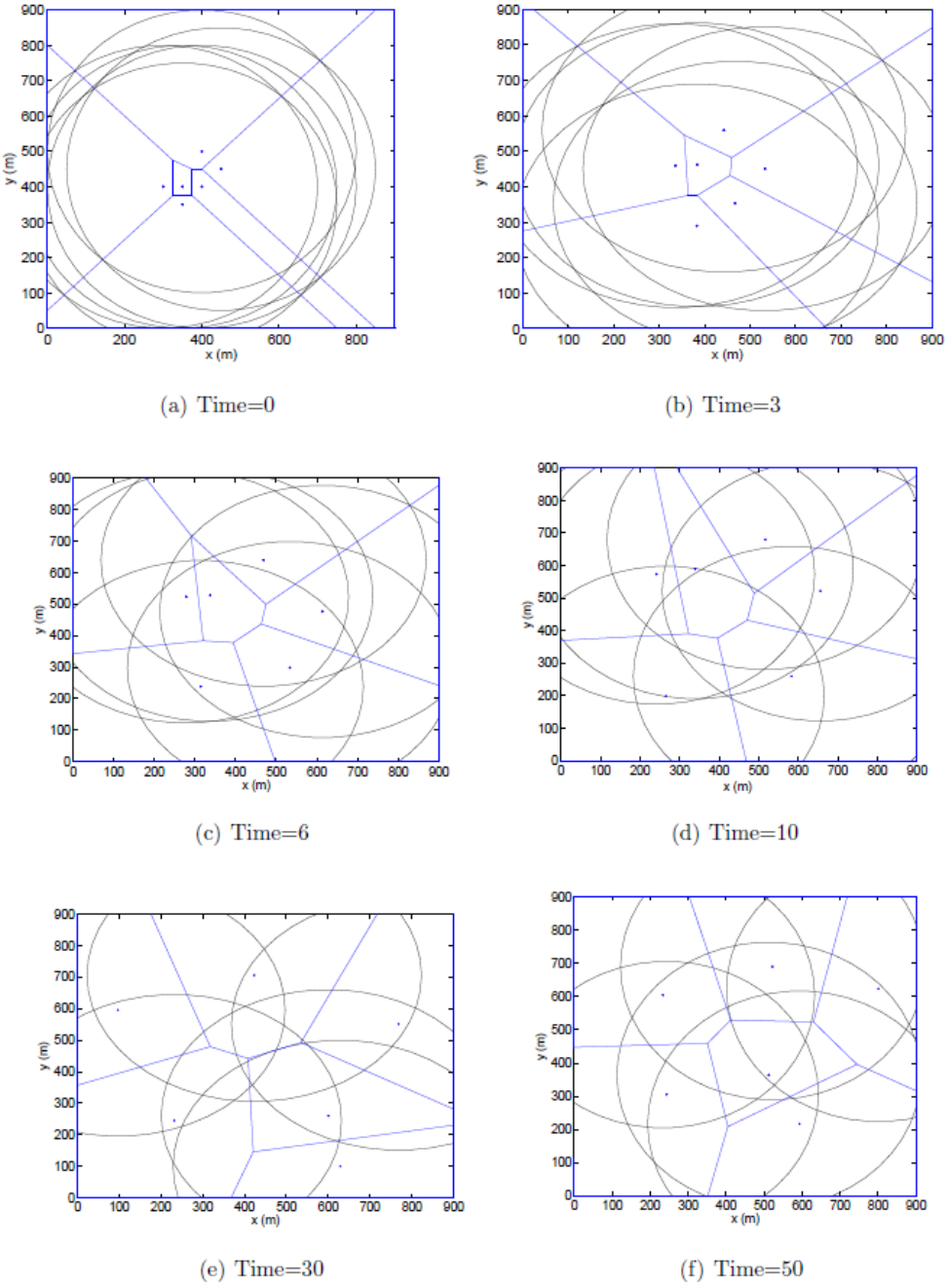


Fig. 1 Distributed air-net coverage control of multiple UAVs

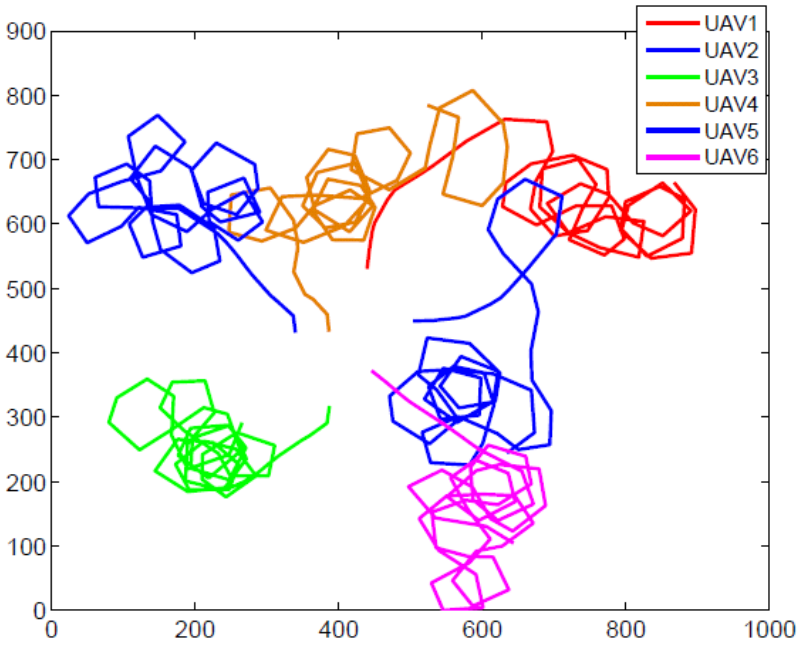


Fig. 2 Trajectory of fixed-wing air-net UAVs

5.5 Mini-toolbox and development / test environment

Matlab toolboxes for the algorithms are attached.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is the initialization section!!
% In this section, we can define the boundary of the map; the number of
% UAVs; the initial positions of UAVs; the speed of UAVs;
% cell size; communication radius Rc; sensing radius Rs;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


UAVposition=rand(6,2)*0.5-0.5/2;
InitialDes = UAVposition;
UAVnum = size(UAVposition,1);

RegionBoundminx = -1;
RegionBoundmaxx = 1;
RegionBoundminy = -1;
RegionBoundmaxy = 1;
TimeSteps = 500;
VR = 0.1;
Detination = zeros(UAVnum,3,TimeSteps);
theta =0: 2*pi/50: 2*pi;
Detination(:,1:2,1)=UAVposition;

CellSize = 0.02;
CellRowNum = (RegionBoundmaxy-RegionBoundminy)/CellSize;
CellColNum = (RegionBoundmaxx-RegionBoundminx)/CellSize;
Rs=0.6;
Rc=1;
DesDegree=1;
StayinRegion = zeros(UAVnum,1);
```

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is the main part of the algorithm!
% This part including the neighbors finding, coverage control protocol
% and also connectivity maintainess
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for Time=1:TimeSteps-1


   Alldistance = zeros(UAVnum,UAVnum);
    for i=1:UAVnum-1
        for j=i:UAVnum
            Alldistance(i,j)=norm(UAVposition(i,:)-UAVposition(j,:));
        end
    end
    Alldistance=Alldistance+Alldistance';


    for i=1:UAVnum
        if abs(UAVposition(i,1))<1 && abs(UAVposition(i,2))<1
            StayinRegion(i)=1;
        end
    end


    AllPoints = {};


    x=UAVposition(:,1);
    y=UAVposition(:,2);

    for i=1: UAVnum
        t=1;
        Aug1=[];
        RefelectUAV=[];
        neighborIndex(i) = {i};
        if StayinRegion(i)==1
            RefelectUAV(1+(t-1)*4,:)=[2*RegionBoundminx-UAVposition(i,1),
            RefelectUAV(2+(t-1)*4,:)=[2*RegionBoundmaxx-UAVposition(i,1),
```

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This part is used to plot the figures. A movie can also be produced, so that
% we can see the UAVs' movement for distributed coverage control
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxRadius = max(Detination(:,3,TimeSteps))/2;
Radius = Rs;

pointbound = [-1,1;-1,-1;1,-1];

% aviobj=avifile('Deployment.avi','fps',2);%¶¨ÒåÒ»¸öaviÎÄ¼þ£¬
figure
Radius = Rs;
CircleX = Detination(1,1,1)+Radius*cos(theta);
                           ta);
 The value assigned to variable 'CircleX' might be unused.
eval(['hh=plot(CircleX',',',',CircleY',',',',''''k-''',');']);
hold on
Radius = Detination(1,3,2)/2;
CircleX = Detination(1,1,1)+Radius*cos(theta);
CircleY = Detination(1,2,1)+Radius*sin(theta);
eval(['hh=plot(CircleX',',',',CircleY',',',',''''r-''',');']);
hold on
for i=2: UAVnum
    Radius = Rs;
    CircleX = Detination(i,1,1)+Radius*cos(theta);
    CircleY = Detination(i,2,1)+Radius*sin(theta);
    plot(CircleX,CircleY,'k -');
    hold on;
    Radius = Detination(i,3,2)/2;
    CircleX = Detination(i,1,1)+Radius*cos(theta);
    CircleY = Detination(i,2,1)+Radius*sin(theta);
    plot(CircleX,CircleY,'r -');
    hold on;
end
voronoi(Detination(:,1,1),Detination(:,2,1));
hold on
plot(pointbound(:,1),pointbound(:,2));

axis([-3,RegionBoundmaxx,-3,RegionBoundmaxy]);

hold off
```

5.6 Open research questions

The following problems remain open at the end of the project and can be considered for future work:

- Time-optimal air-net coverage problem
- 3D coverage problem

References:

[1] Luciano C. A. Pimenta and Mac Schwager, et.al, "Simultaneous Coverage and Tracking (SCAT) of Moving Targets with Robot Networks", WAFR'08, pp. 85-99.
[2] M. Schwager, B. Julian, M. Angermann and D. Rus, “Eyes in the Sky: Decentralized Control for the Deployment of Robotic Camera Networks,” Proceedings of the IEEE, September, 2011, vol. 99, no. 9, pp. 1541-1561.
[3] M. Schwager, D. Rus and J. J. Slotine, “Decentralized, Adaptive Coverage Control for Networked Robots,” International Journal of Robotics Research, March, 2009, vol. 28, no. 3, pp. 357-375.