# A PETRI NET BASED SIMULATION FOR MULTIPLE UNMANNED AERIAL VEHICLES

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Puneet Mehta

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2018

Fargo, North Dakota

North Dakota State University

Graduate School

**Title**

A Petri Net Based Simulation for Multiple Unmanned Aerial Vehicles

**By**

Puneet Mehta

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard

Chair

Dr. Brian M. Slator

Dr. Ross Collins

Approved:

| | |
|---|---|
| 01/15/2019 | Dr. Kendall Nygard |
| Date | Department Chair |

## ABSTRACT

Nowadays more and more Unmanned Aerial Vehicles (UAVs) are being mass produced and are being used for a lot of activities like exploring never before explored areas of the world, recreational work, rescue missions etc. In today’s world violence and attacks have increased and to have personnel in such extreme places will always result in loss of life. To help reduce the loss of lives in situations like this we can use these Unmanned Aerial Vehicles to sweep the affected areas and also attack identified targets.

The objective of my study is to create a simulation where multiple UAVs can perform a sweeping search and attack targets. This is a stepping-stone in the development of more unique ways in which we can use Unmanned Aerial Vehicles to save lives and to move towards the future of working side by sides with humans.

## ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Kendall Nygard for his support and guidance. My research would not have been accomplished if I didn't have Dr. Nygard helping me all the way. Dr. Nygard has been available to help whenever I needed help with the problems over the course of my Masters. Not only his expert comments and suggestions helped motivated me to improve my research and complete it, but his patience and kindness that he showed me has enabled me to achieve what I have achieved today. Your expertise and knowledge has inspired me a lot. Thank you Dr. Nygard for all your time and effort and also guiding me in the right direction.

I would also like to thank my committee members, Dr. Brian M. Slator and Dr. Ross Collins for taking out the time to be a member of my supervisory committee. I really appreciate your kindness and thank you for your time. I am also thankful to all the faculty members and staff who have always been there to help and provide support to a student like me.

Finally, I would like to thank my parents Satinder Mehta and Shail Mehta for all the love, support they have showed me and motivated me from the first day of my very first school to the very last day of my Master's program. Last but not the least I would like to thank all my friends for being there when I needed them the most. Thank you for having the faith and trust in me.

## DEDICATION

This paper is dedicated to my parents **Satinder Mehta** and **Shail Mehta**, and also to my brother **Punkit Mehta**. You all have been there every time I needed you, no matter how tough the situations were. Thank you all for your love and support throughout my journey.

## TABLE OF CONTENTS

# LIST OF FIGURES

# 1. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) have become an integral part of society; they are being used as tools for entertainment, search and rescue, as well as exploration. Almost every industry is now dependent on Unmanned Aerial Vehicles not only because of their agility, speed and maneuverability, but also for the fact that they are easily programmable and relatively inexpensive. As technology improves so will the use of these UAVs, as we have already seen UAVs being used for cargo delivery, cinematography, journalism, agriculture and the military. Whatever the industry, they have proven to be successful in every field.

UAVs are classified into several categories depending upon their use, shape, structural strength, and electrical efficiency as well as aerial mechanics. Despite several divisions and parameters for UAVs, their generalized model and working concept has not changed a lot. The assumption can be made that almost every UAV works on the same principle. Some types of aerial vehicles require the use of remote controller for their navigation and motion planning. Such aerial vehicles are also classified under the banner of UAVs because they do not have any human driver. The input received from the intelligent system of UAV is used to control it in a better way.

However, the UAVs are still very primitive in their applications around the world. There is still a long way to go before they can be completely utilized. Currently, Unmanned Aerial Vehicles are of two types:

- Handled by human controllers, and
- Autonomous

Between these two types of UAVs, the latter has been the most primitive. UAVs that are handled by human controllers are much more advanced, easily produced and are used by many industries, especially the military, which have used UAVs to fight terror and reduce casualties. For proper use, handheld UAVs have to be in range of the controllers and require a human to guide them, which makes them risky for military personnel.

The integral part of performing a strike is having coordination and communication between military forces and the UAVs. Based on several studies, Petri Nets can help in developing those communications and coordination between multiple UAVs. K Jensen [5] informs us about the basic concepts, analysis methods and practical use of colored Petri Nets. The use of Petri Nets in the development of UAVs can be beneficial as explained by K Jensen [19] in his paper using colored Petri Nets for developing simulation.

Communication, coordination, and effective search methods can lead military personnel to targets much faster. One such search method is the sweeping search method, which allows several UAVs to sweep given areas even if the targets are moving. According to a study by Alfred M Bruckstein [6, 7]**,** one method of hunting a moving target is a swarm of UAVs, which uses different flying patterns and large sensors to track and attack moving targets.

By conducting sweeping searches with UAVs, this technology can help save a lot of human lives. The agility and the speed of the UAVs would give militaries an upper hand in warzones or impeding attacks.

In addition, communication between several UAVs using the colored Petri Nets can also prove important in search and rescue missions after a major disaster, such as large-scale floods and earthquakes. The uses of UAVs are only limited to one's imagination.

For UAVs, the targets can range from a single terror camp, vast terror organizations, or to enemies that are constantly on the move. To track the movements of specific targets, the UAVs can be fitted with GPS trackers, attached with sensors, and can even get real time data from the navigation satellites to help them navigate through rough terrains and detect obstacles in their path.

Unfortunately, smaller UAVs cannot handle carrying heavy sensors. Research was conducted by Sanjiv Singh [4]**,** to make the UAVs fully automated where they can detect obstacles and change their path mid-flight. Fitting UAVs with small cameras and using software algorithms can replace heavy sensors.

This paper mainly focuses on three concepts, considered to be the building blocks of this simulation. Based on a paper by Kendall Nygard, et al. [1, 2, 3] the three things that the simulation uses and are described in the paper are:

- UAVs
- Sweeping Search
- Targets

The simulation creates a visual representation of UAVs in a confined environment, conducting a sweeping search using algorithms to navigate in a straight line, and to find the desired number of targets that are constantly moving in the same space. The UAVs in the simulation track down the targets and attack them; the simulation goes on until all the targets have been destroyed.

This paper is divided into chapters and remainders of the chapters are as follows. Chapter 2 of the paper describes the details of previous studies that have been conducted on the UAVs and the studies that are related to this paper. Chapter 3 describes how the Petri Nets influenced the build of the simulator. Chapter 4 describes how the simulator was build and what its components are. Chapter 5 describes the suggestions of future work that can be implemented in this simulation to make it more autonomous and useful, using tools and programs that are already available. Chapter 6 contains the conclusion for this paper.

# 2. BACKGROUND AND RELATED WORK

Ever increasing terror organizations in every part of the world has led to an increased loss of human life, which has become a pressing issue worldwide; finding a solution has become a necessity. Building this simulator is the first step in making autonomous UAVs that can attack any given target.

Except the attacking purposes, it can also serve as a better navigator for an instance, a better-organized and intelligent UAV system can reach to a place where a normal human being or other mobile robots cannot reach due to the limitations. In case of disaster or in case of emergency, UAVs can be used to navigate through required task and report the results of its navigation. Reported results can be further use to improve the quality of operation going on.

This paper is based on Kendall Nygard's et al. [1, 2, 3] work on Petri Nets based UAV simulation. This UAV simulation is a visualization of a sweeping search attack on several targets.

Using one such study that describes the agent-based framework of UAVs by Kendall Nygard et al. [1, 2, 3] allows us to design a framework for the simulation. In this study, two approaches were introduced to build a better simulator. These are:

- Bottom control using swarm behaviors
- Linear Programming approach

These two ideas help add or remove parts of the program in the simulator without changing the whole system every time. The UAVs simulator needs different integrations of features to be compatible, such as a clock or log book that records the reports. Applying approaches from this study helps to make this process less complex.

Another study conducted by Kendall Nygard et al. [1, 2, 3] elaborates on the use of the two necessary components that have a bigger impact on the simulator, which helps guide the simulator in performing the tasks assigned to it.

Those two things are:

- Colored Petri Nets
- Sweeping search algorithm

Sweeping search algorithm has been used in the simulator to help guide the UAVs in a confined space while in a **swarm** formation to cover a larger area and find moving targets easily. Likewise, Colored Petri Nets will be the mainstay of the simulator to design the architecture of the UAVs control structures. With the help of these control structures, building multiple UAV simulations will be easy. These things applied with other features in the simulator make it better for visualization.

Figure **[**1] shows the use case model for the simulator.

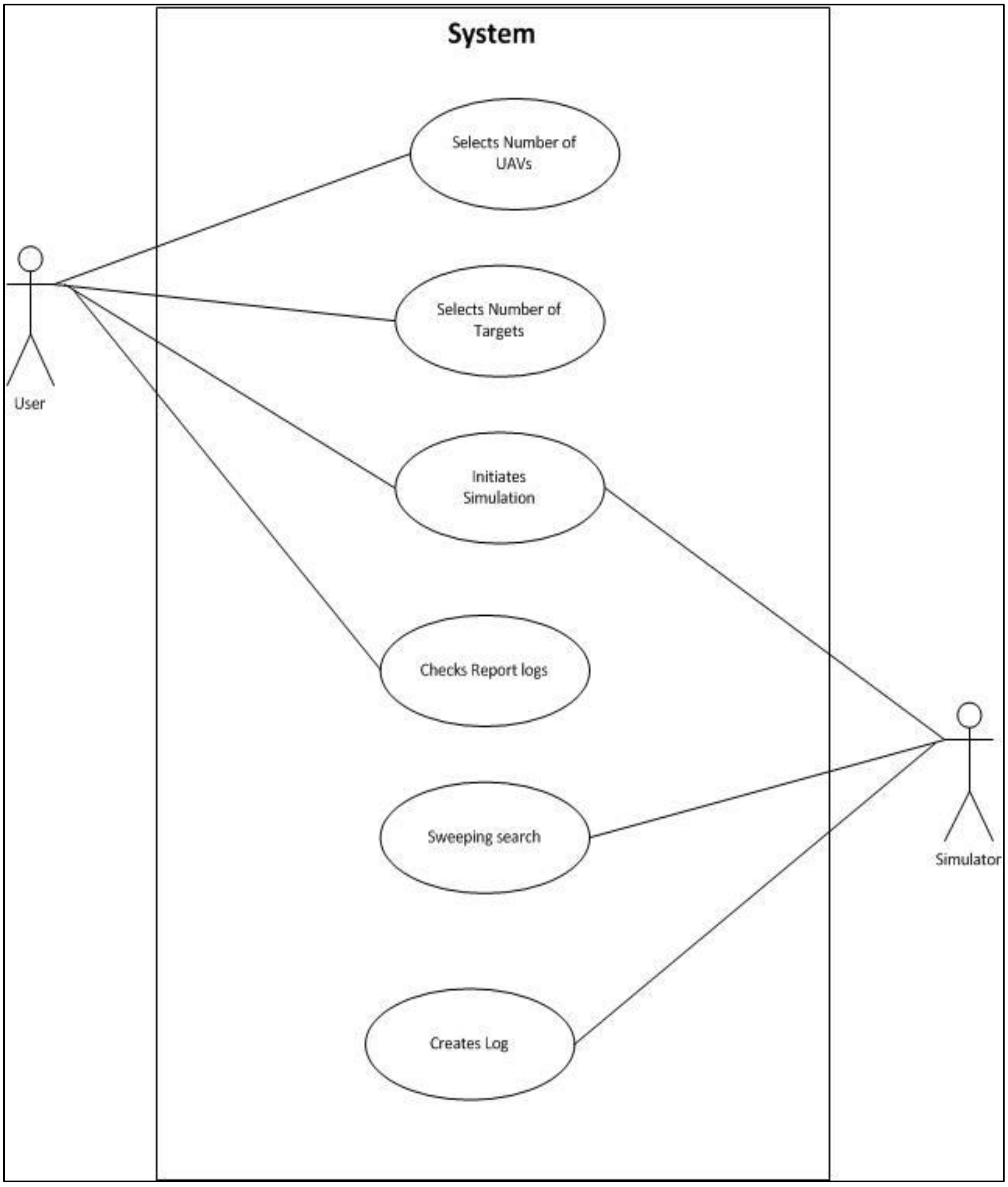


Figure 1: Use Case Diagram

A study conducted by Sanjiv Singh [4]**,** describes the use of multi robot search that could help solve the problem of locating a moving target. This study, like many others, uses the same concept of applying multiple UAVs (Swarm method) to search for a given target in a known or controlled environment. Not only that, but the study tries to assume that the environment is well known as it calculates a path that would most likely intersect with the targets.

Accordingly, a searcher path makes this study different than the sweeping search algorithm used in the simulator, as the latter searches for the whole area or the known environment no matter what, until it finds the moving targets.

There are other studies that look in to building their own systems that use several UAVs to attack a single static, non-moving target or multiple moving targets. Some use UAVs for surveying using cameras, and some are using UAVs to attack a specific target. A study conducted by Kendall Nygard et al. [1, 2, 3] tries to use multiple UAVs carrying explosive materials to attack a single target head on, so that the attack is successful.

According to Liu Qiang [10] the UAV swarm behavior is a very complex system to build but once built can also produce emergent behavior. The swarm behavior has been directly taken from the social animals in the nature, like the flock of Auklet or the fish school. The swarming behavior has its benefits and when used in the simulation for the UAVs movements reaps greater rewards. When it comes to the search and rescue or attacking a target to get more accurate result this particular behavior is very helpful. A form of these behaviors is incorporated in the simulator where the UAVs are in a formation while sweeping through the given field in the simulator.

The other related work around the swarm behavior comes from Argel Bandala [11], where he discusses about how the swarm behavior can be used for the formation purposes of the UAV simulation. According to this paper swarming behavior is often derived from a group or some social behavior of animals or insects, but this paper parleys about the use of aggregation, social foraging and flight formation to come up with the swarming behavior of the UAVs.

The swarm behavior has been used in many simulation across many industries. The swarm behavior could have been used in this simulation as well, but the UAVs in this simulation use the formation flight behavior as mentioned above. The formation flight behavior is the a form which the UAVs maintain throughout the run time of the simulation, by which it means is the UAVs remain in the same formation when moving from point a to point b and then moving from point b to a without losing their formation. According to the paper by Luo, Delin [10], the UAVs need to keep the formation to be effective as well as be capable enough to change the formation in the times of environmental change etc. and that is why a control strategy is needed to be designed for the transformation of the formation.

The swarm behaviors of the UAV can be unpredictable and sometimes lead to undesirable behaviors known as the emergent behaviors. According to the paper by Shweta Singh [12], these actions emerge when the entities or the agents interrelate with the environment making more complex vigorous behaviors. The change in the system due to the change in the environment which it comes in contact with leads to the emergent systems. These types of behaviors or actions can come in the favor of the UAVs in the simulation, where the targets are randomly moving all the time and applying these types of emergent behaviors in the UAVs can help the UAVs to tackle the situations where the targets randomly move or when they are faced with obstacles and need to change the direction or communicate that information to the nearby UAVs.

All these related studies that are being conducted, can be used for the betterment of the simulator, by adding the swarming behaviors or using the flight formation behaviors or by simply using the emergent systems it can be easily said that the simulator or the UAVs can actually achieve what is required of them. The next chapter describes the use of one such program called the Petri Nets to enhance the communication between the UAVs and help them navigate through the different terrains.

# 3. PETRI NETS

Petri Nets is a modeling language that is used0 for the distributed systems. According to Kendall Nygard et al. [1, 2, 3], paper in which he debates about the use of Petri Nets in the simulation to create the communication channels. These channels not only help the UAVs send information about the paths, obstacle and nearby targets but also send that information back to the server or the command center where the UAVs were launched. The most commonly used type of Petri Nets is called the Colored Petri Nets.

Colored Petri Nets are also termed as CPNs or CPNets. It is a basic type of graphical processing language that is used to construct the models via color processed methodologies so that they can be analyzed at several situation and boundary conditions. It is often classified as functional type of programming language as well. It uses the events as discrete objects during modeling so that discrete outputs can be obtained for decision-making processes. It can be used to model the general operations or specific operations depending upon the situation it is relevant in. In this case the processing would be made for UAVs so that they can be used for navigation and motion planning.

Colored Petri Nets can be used for modeling of algorithms that are distributive in nature, data networking, and numerous protocols of communication. The control architecture required for the navigational purposes of UAVs required the modeling of concurrent as well as communication systems and color petri nets models are completely able to provide such characteristics. Colored Petri Nets models are also being utilized by the manufacturing industry, multi agent controlling systems, business operations as well as various industrial applications.

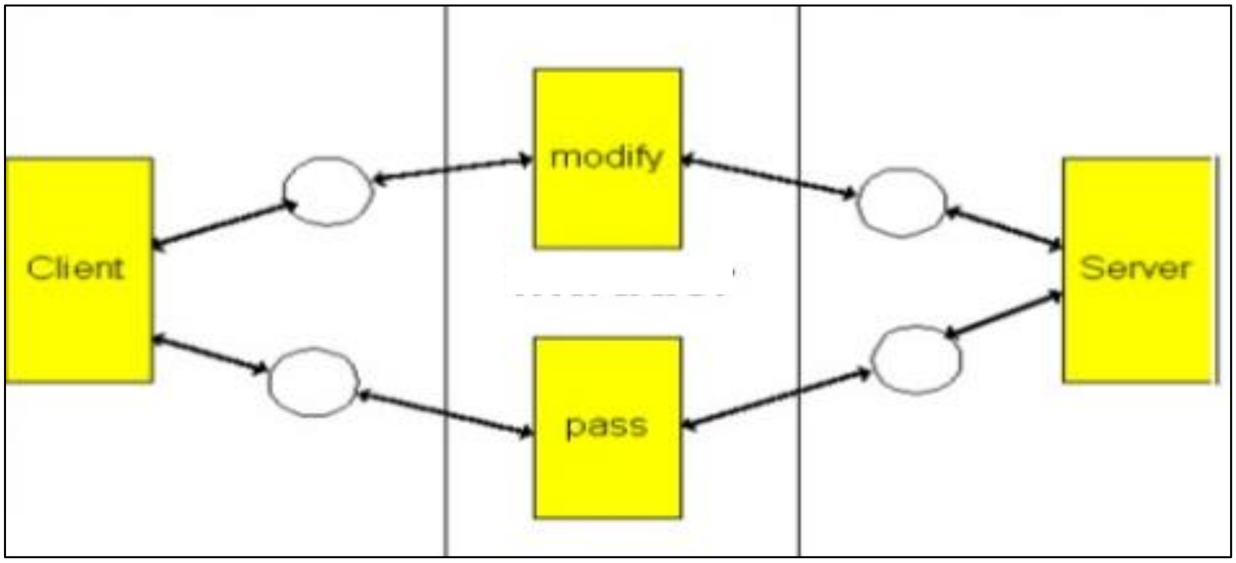


Figure 2: Colored Petri Nets Communication System Model

A simplest model of the communication system via Color Petri Nets methodology has been shown in above figure. In this example a client (UAVs for this case) is communicating via two-channel intruder to a server (Control system). This generalized communication model can be used to control the UAVs system in a more sophisticated way, if they are adopted for this application.

Colored Petri Net based system is highly capable of informing the system's start as well as the transitions made by the system, which is often termed as events. The simulations of a model based on CPN makes it possible to establish and investigate numerous cases so that the behavior and response of the modeled system can be explored. It is a common practice in the simulations being used for debugging as well as investigation of the design of any system. It is the intrinsic property of colored petri nets programming that they can be easily simulated in an interactive or totally autonomous way. Interactive CPN simulations cover the technical areas in which a one-step compilation debugging option is utilized. A walk over from entire model's point of view can be captured using interactive simulations mechanisms of CPNs. The expectations of model based on CPNs can also be verified and validated using the interactive simulations mechanism.

It is to be noted that, when the interactive mechanism based simulations are in progress the modeler unit is basically in charge and it is also responsible for determination of the subsequent steps after the selection of events that are currently enabled. The effects produced by the independent steps can possibly be observed directly in the form of a graphical map. The map resembles the hierarchy of CPN model, but it includes the details of effects made by each event in the model. Another type of simulation mechanism that was named before in this study was autonomous simulation. The discussion ahead would involve the description of autonomous mechanisms based simulation for CPNs. This type of simulation is reasonably compared to the program execution.

The requirement for such simulations is to fulfill the rapid decision making demands so that the computations cost can be reduced by as much order as possible. Testing of any CPN model as well as the analysis of its performance can only be obtained via autonomous simulations instead of interactive simulations, which provides details about the individual event, but not the entire program or model. The modeler units in this type of simulations are used to feed in the breaking as well as stopping criteria. Obtaining data that is concerned and related to the system's performance can do analysis of performance for the model. Virtual data loggers are used in this scenario on the model map so that data can be collected in required format.

Timings of the modeled events are an important aspect in determining model efficiency. In fact the correctness of model is dependent on the timing of each event of the corresponding model. Due to the effect of time taken by events on the correctness of model the decisions of different design possibilities of models are altered. CPN models are fully capable of capturing the functioning of events in timely manner so that the correctness of the model should be compiled in real time. The timing ability of CPNs also makes this model able to establish and investigate the performance analysis in a better way.

Numerous measures of performance such as, lengths of queue in any system, delays and validations of the real time executing models can be easily checked by CPNs. In short, CPNs would be able to provide quite robust computer architecture when it would be used under UAVs for the navigational or motion planning purposes.

Another ability of CPNs involves their structuring into different group of modules so that the large specifications can be handled easily. The grouped modules can easily interact among themselves via predefined interfaces. This characteristic is quite similar to the one used in programming languages, especially in C#, thus the system is also quite easy to maintain and develop in C sharp scripting language. The structuring in-group modules resemble the mechanism of hierarchical divisions that allows sub modules in a module as well.

CPN models are animated by a technique termed as visualization. In visualization quite heavy graphical data is used for animation purposes, the animation of CPN models is quite similar or related to its simulations, but still there is a fundamental difference that for animation only one case is simulate throughout the entire model whereas in simulations multiple cases can be observed at any interval of time and in any sequence.

One of the important aspects of visualization technique is the presentations of possible ideas about the design as well as the analysis of results by the use of the domain concepts of the running application/model. It is quite an important and most widely used aspect of using visualization technique with CPNs especially for UAVs because they need continuous navigation and tracking. It also helps in understanding of the model to those who are not familiar with CPN based models. Now that we have discussed the importance and definition of visualization technique let us discuss the methods to use this technique.

There are multiple methods of using domain-based graphics at the top of a CPN model for visualization.  The use of messaged based sequential charts, which are often, termed as sequence diagrams as well for the visualization of the exchanges in messages between the communication protocols. The visualization model of a CPN based model for its use in communication protocol that can be used in UAV.

Further review of the literature available about CPN based models states that it is quite formal in its operations. The CPN based models are established via a language called CPN model language and the language possess proper semantics as well as syntax based on its computational strategy.

These qualities of CPN based models verify the properties of any system they are undergoing through, as in this case its UAVs. The verification of properties would include the check on the matter that whether certain properties that are required to be working properly in the system are doing their work or not.

State space methodology of computer architecture is employed in this regard to verify the properties of system under discussion. It is to be noted that the state spaces are used to compute all of the available and reachable stats. Then it computes the changes in the states for the CPN model and then represents the changings in the form of a graph directed by the changes in states. The nodes in the graph are representations of the states while the arcs in graph are representations of ongoing events.

Another feature of state space is that it can be easily constructed for a given model. The benefit of constructing state spaces lies in the fact that it reduces the computational cost because it can verify very large set of instructions using the nodes it has already established. To conclude the discussion about review of state space methodologies and their relation with CPN models solution it is stated that the literature evidenced the validation of functional efficiency of any system model by CPN or timed CPN can be done by the use of state space methodology.

The modeling programming language of CPNs must be learned before digging deeper into its use. The practical use of CPNs cannot be achieved once the basis of the syntax as well as semantics of its working framework isn't understood already. The programming language for CPNs is analogues to the available programming languages, but it is closely related to C# and thus the model for the simulations of UAVs in this study would also be constructed using the C# scripting tool. It would help in modeling of the system and also the redundancy in the model as well as the learning phase would be reduced up to reasonable extent.

Except CPN model there is another technique available in the literature to conduct the similar tasks in a much different way. The technique is named as sweeping search algorithm. The technology of sweeping search algorithm is quite different than CPNs and in comparison CPNs are much more advanced.

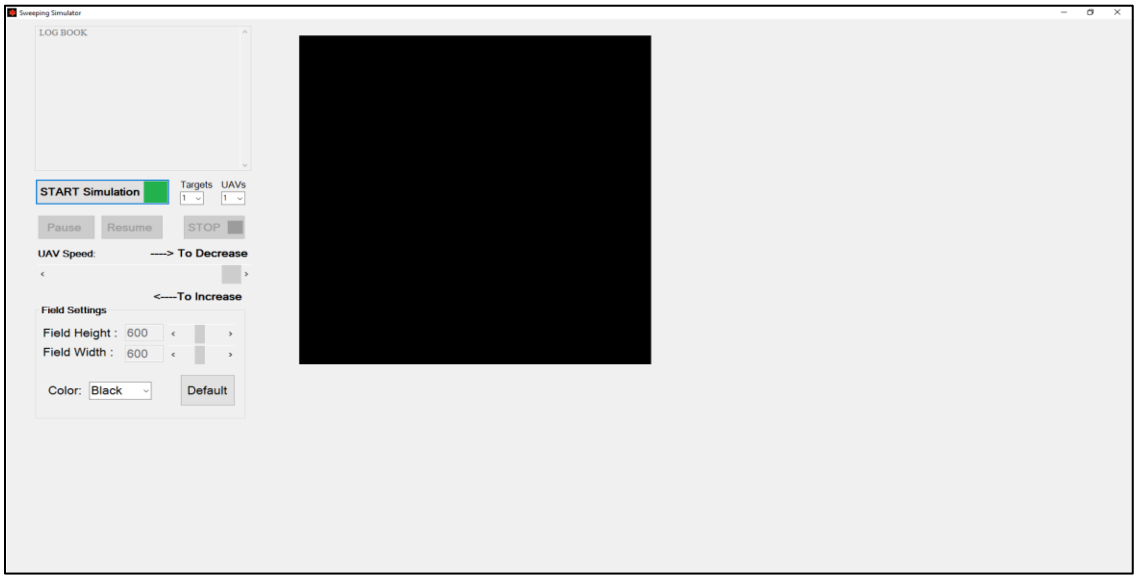
It is an important algorithm in many UAVs navigational methodologies and has been adopted by many good researchers, but the flexibility and performance provided by the Colored Petri Nets models is way more than these algorithms. The requirement of skills is also quite high in such algorithms.

There have also been many studies regarding building UAVs that have one feature or another. One study conducted by Alfred M Bruckstein et al. [6, 7], looks at swarm UAV algorithms to solve the search of smart targets. The swarm UAV algorithm is much more efficient in navigating and locating targets which are constantly moving as they can cover a lot of area. Using the swarm algorithm increases the success rate of UAV attacks.

Furthermore, we can describe all the uses of the Colored Petri Nets and how if they are put to use in the simulator can improve the overall use of the simulator as well as give the UAVs a big improvement from the rest of the UAVs that are being used around the world for the same purpose. In the next chapter, we discuss about how the simulator was built and what all goes in the front end and the backend of the simulator, also I will include the methods and the constructors as well as images of the actual simulator that I have built in the next chapter.

# 4. THE BUILD

This section describes in detail about how the simulator was built.



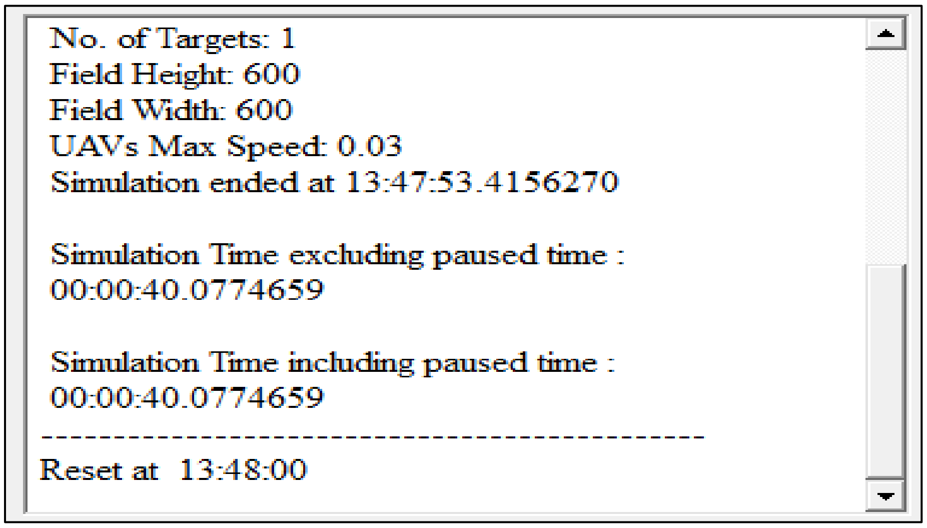
Targets: 1
UAVs: 1
Field Height : 600
Field Width : 600
Color: Black

Figure 3: The Simulator

<u>The Front End:</u>

As soon as the application is launched, a form is displayed in full screen. The form contains:

- Log book



```
No. of Targets: 1
Field Height: 600
Field Width: 600
UAVs Max Speed: 0.03
Simulation ended at 13:47:53.4156270

Simulation Time excluding paused time :
00:00:40.0774659

Simulation Time including paused time :
00:00:40.0774659
-----------------------------------------------
Reset at  13:48:00
```

Figure 4: Log Book

- Start button

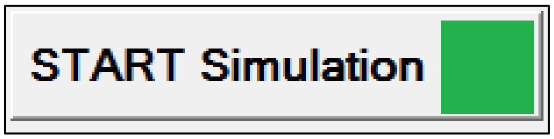


Figure 5: Start Button

- Drop down for targets and UAVs

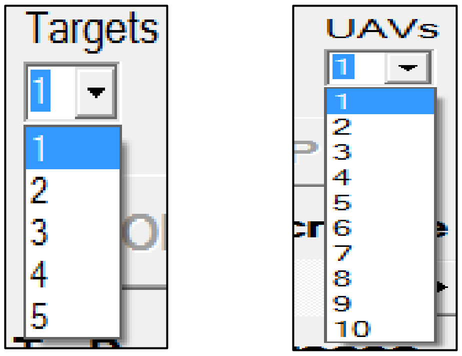


Figure 6: Drop Down Buttons for Target and UAVs

- Pause button

Figure 7: Pause Button

- Resume button

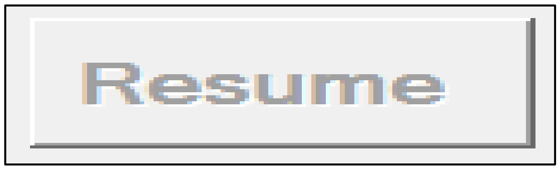


Figure 8: Resume Button

- Stop button

Figure 9: Stop Button

- Scroll bar to increase or decrease the UAVs speed

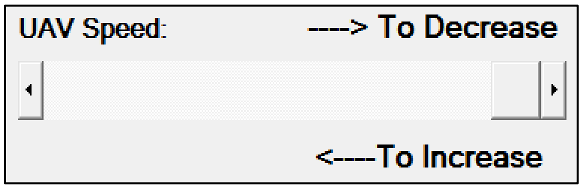
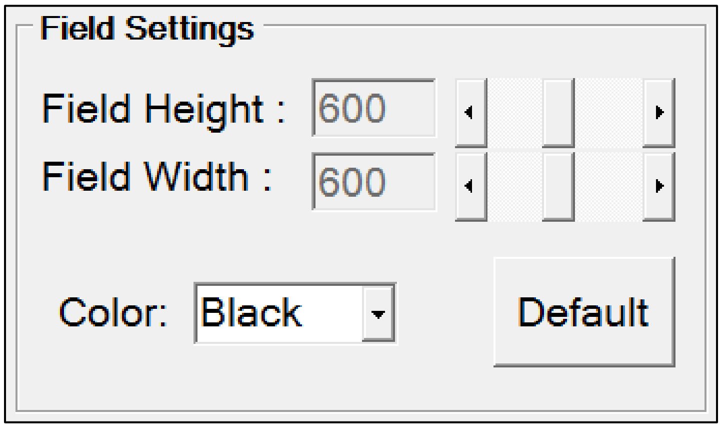


Figure 10: UAV Speed Scroll Button

- Field Settings
    - Scroll bars to increase and decrease the height and width of the screen
    - Drop down to pick a color for the screen
    - Default button to take the screen back to the default size


Field Settings

Field Height : 600

Field Width : 600

Color: Black

Figure 11: Field Settings

- Black default screen.

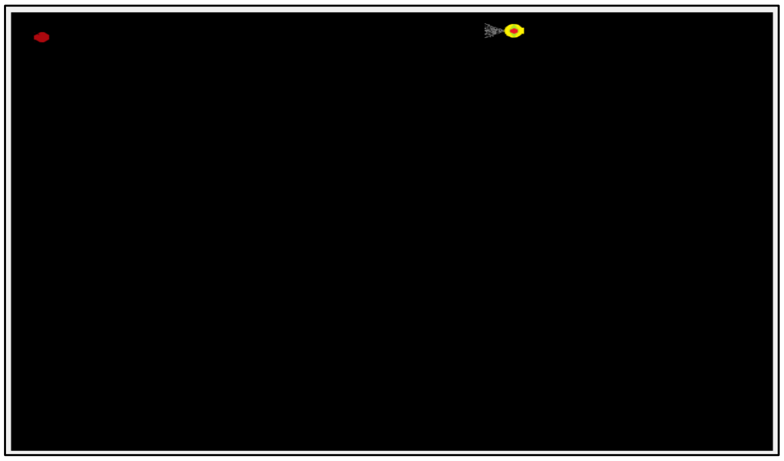


Figure 12: Simulator Screen

The Backend:

On the backend the simulator consists of:

- Program Class
- Form Class
- Drones Class
- Target Class
- UAV Class
- Logbook Class

Program Class:

This is the class holding the main method of the application invoking the Form class.

```
static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
```

Form Class:

The form class holds all the methods and variables.

Class Variables:

The form class has some class variables for number of targets, number of UAVs, user defined class logbook's object, two object arrays are declared using abstract class, Drones for UAVs and Targets. Other class variables are used as flags for example: pause flag, field height and width, kill check flag etc.

```
public partial class Form1 : Form
    {
        int pause = 0; // pause flag
        int numOfTargets;
        int numOfUavs;
        LogBook logs;
        Drones[] uavs;
        Drones[] targets;
        double speed;
        double maxSpeed;
        int killCheck;
        int field_height;
        int field_width;
        int[] blast_flag ;
        String colour;
        Thread targetsMove;
        int blastdelay;
        Random rand;
```

Constructor:

The code in the constructor runs immediately when the Form class is invoked. It initializes all the components in the form, sets the default speed and max speed for speed changer functionality, it also sets default background color of the field, and disables resume, pause and stop buttons. It also maximizes the window size.

```
public Form1()
        {
            InitializeComponent();
            speed = 100 * 0.0003;
            maxSpeed = speed;
            colour = "Black";
            rand = new Random();
            button1.Enabled = false;
            btnResume.Enabled = false;
            resetBtn.Enabled = false;
            blast_flag = new int[]{ 0, 0, 0, 0, 0, 0 };
            this.WindowState = FormWindowState.Maximized;
        }
```

Form Load Method:

The form's load method runs after the form has been initialized. This method initializes time ticker event that loops to keep the drones moving, field paint event, field height, width values and scroll bar.

The ticker event initializes a method called timer_Tick which runs in loop moving the UAVs, checking for killed targets and logging the events.

Field paint event calls a method called Form1_Paint which uses the field width, height and color class variables to paint the field.

```
private void Form1_Load(object sender, EventArgs e)
    {
        timer.Interval = 1;
        timer.Tick += new EventHandler(this.timer_Tick);
        Paint += new PaintEventHandler(Form1_Paint);
        field_height = 600;
        field_width = 600;
        this.Invalidate();
        fieldHeightTextBox.Text = field_height.ToString();
        fieldWidthTextBox.Text = field_width.ToString();
        heightScrollBar.Value = field_height / 15;
        widthScrollBar1.Value = field_width / 15;
    }
```

StartBtn_Click Method:

When start button is clicked this method is called and the first thing it does is suspends the form from any user changes. It takes the selected values from the form, values like number of targets and UAVs. It then disables some of the components like dropdowns to select number of drones, start button, field settings, log book, but it also enables some components like pause button, resume button, stop button and two object arrays were declared using abstract class.

It also initializes drones in class variables as UAVs and Targets, these are the two user defined classes inheriting the Drones abstract class. The size of the array is declared by the selected values from the form. A method is called to generate the drones/sprites. Log book instance is initialized.

Two time tickers are started; one for moving the targets and one to move UAVs. This is done because targets have different speed than UAVs. And finally, layout is resumed, and form is updated and refreshed.

```
private void startBtn_Click(object sender, EventArgs e)
        {
            this.SuspendLayout();
            numOfTargets = Convert.ToInt32(NumTargets.SelectedItem);
            numOfUavs = Convert.ToInt32(NumUav.SelectedItem);
            NumTargets.Enabled = false;
            NumUav.Enabled = false;
            button1.Enabled = true;
            btnResume.Enabled = true;
            resetBtn.Enabled = true;
            uavs = new Uav[numOfUavs];
            targets = new Target[numOfTargets];
            generateSprites();
            startBtn.Enabled = false;
            FieldSettings.Enabled = false;
            defaultBtn.Enabled = false;
            logs = new LogBook(tbLog);
            logs.startTime(numOfTargets, numOfUavs);
            this.DoubleBuffered = true;
            timer.Enabled = true;
            tbLog.Enabled = false;
            pause = 0; // update pause status as false
            targetsMove = new Thread(moveTargets);
            targetsMove.Start();
            timer.Start();
            this.ResumeLayout(false);
            this.Invalidate();
            this.Update();
            this.Refresh();
        }
```

## GenerateSprites Method:

This method calls two different methods to generate specific number of targets and UAVs.

```
private void generateSprites()  {

            generateUavs();
            generateTargets();
        }
        private void generateUavs(){
            int u_limit = 25;
            int x, y;
            for (int i = 0; i < numOfUavs; i++)
            {
                x = 460;
                if (i == 0)
                    y = 30;
                else
                    y = u_limit + uavs[i - 1].getYCoordinates();
                uavs[i] = new Uav(x, y, this);
            }
        }
        private void generateTargets()
        {
            for (int i = 0; i < numOfTargets; i++)
            {
                targets[i] = new Target(field_width + 500, field_height + 30, this);
            }

        }
```

MoveUav Method:

This method is invoked from time_ticker method, which in turn calls the move method of the UAV class to move the UAVs to a specific direction.

```
private void moveUav()
        {
            for (int i = 0; i < numOfUavs; i++)
            {
                uavs[i].move(field_height+30, field_width+500);
                if(i == Uav.count -1 && Uav.directionChangeFlag == 1)
                    Uav.directionChangeFlag = 0;
            }
        }
```

Timer1_Tick Method:

Start button click invokes this method and this method calls a move method of the target class with the field parameters to move all the targets in random direction. This method runs until the timer is paused or stopped.

```
private void timer1_Tick(object sender, EventArgs e)
        {
            this.Invoke((MethodInvoker)delegate
            {
                for (int i = 0; i < targets.Length; i++)
                {
                    targets[i].move(field_height + 30, field_width+500);
                }
            });
        }
```

IsTouching Method:

This method is invoked by check_kill method. This method returns a value if two images have same location boundaries.

```
private bool IsTouching(PictureBox p1, PictureBox p2)
    {
        if (p1.Location.X + p1.Width < p2.Location.X)
            return false;
        if (p2.Location.X + p2.Width < p1.Location.X)
            return false;
        if (p1.Location.Y + p1.Height < p2.Location.Y)
            return false;
        if (p2.Location.Y + p2.Height < p1.Location.Y)
            return false;
        return true;
    }
```

## Timer_Tick Method:

Start button click also invokes this method. This method moves the UAVs and checks for kills and goes in loop until stopped, paused or if it's a game over. It calls two separate methods to move the UAV location coordinates by one and to check the kills.

```
private void timer_Tick(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(TimeSpan.FromSeconds(speed));
        if (blastdelay > 15)
        {
            for (int i = 1; i <= numOfTargets; i++)
            {
                switch (i)
                {
                    case 1: blast1.Visible = false; break;
                    case 2: blast2.Visible = false; break;
                    case 3: blast3.Visible = false; break;
                    case 4: blast4.Visible = false; break;
                    case 5: blast5.Visible = false; break;
                }
            }
            blastdelay = 0;
        }
        else if (blastdelay > 0 && blastdelay <= 15)
            blastdelay++;
        Uav.updateRadar(field_width + 500);
        moveUav(); // move UAVs
        check_Kill(); // check if any target is killed
        killCheck = 0;
        for (killCheck = 0; killCheck < targets.Length; killCheck++)
        {
            if (!targets[killCheck].getKillStatus())
                break;
        }
        if (killCheck == Target.count)
        {
            logs.finish(numOfTargets, numOfUavs, field_height, field_width, maxSpeed);
            timer.Stop();
            pause = 1;
            tbLog.Enabled = true;
            button1.Enabled = false;
            btnResume.Enabled = false;
        }
    }
```

## Check_Kill Method:

It is invoked from time_ticker method too, this method checks if any UAV is coming in contact with any target. If the UAV is, then this method generates a blast image for a delay of 1 and then removes the blast image and the target by updating its kill status.

```
private void check_Kill()
    {
        for (int i = 0; i < uavs.Length; i++)
        {
            for (int j = 0; j < targets.Length; j++)
            {
                if (targets[j].getKillStatus() == false)
                {
                    if (IsTouching(uavs[i].getDroneBox(), targets[j].getDroneBox()))
                    {
                        targets[j].setKillStatus(true);
                        tbLog.AppendText("\r\n killed Target " + j + " at" + DateTime.Now.ToString(" HH:mm:ss"));
                        blast1.Location = new Point(targets[j].getXCoordinates(), targets[j].getYCoordinates());
                        blast1.Visible = true;
                        blastdelay = 1;
                    }
                }
            }
        }
    }
```

## TextBox1_TextChanged Method:

This method is invoked if the height of the field is changed in text box in the form. This method checks to see if the height will go out of the screen or not.

```
private void textBox1_TextChanged(object sender, EventArgs e) {
        int field;
        if (!int.TryParse(fieldHeightTextBox.Text, out field))
            field_height = 600;
        else {
            if (field > 0 && field <= Screen.PrimaryScreen.Bounds.Height - 100)
                field_height = field;
            else {
                fieldHeightTextBox.Text = "600";
                field_height = 600;
                tbLog.AppendText("\r\n----------------------------------------***");
                tbLog.AppendText("\r\nCannot exceed field Height more than" + (Screen.PrimaryScreen.Bounds.Height -
100) + " or less than 1");
                tbLog.AppendText("\r\n----------------------------------------***");
            }
        }
```

```
    this.Invalidate(); }
```

FieldWidthTextBox_TextChanged Method:

This method is invoked if the width of the field is changed in the text box in form. This method checks to see if the width will go out of the screen or not.

```
private void fieldWidthTextBox_TextChanged(object sender, EventArgs e) {
    int field;
    if (!int.TryParse(fieldWidthTextBox.Text, out field))
       field_width = 600;
    else {
       if (field > 0 && field <= Screen.PrimaryScreen.Bounds.Width - 100)
          field_width = field;
       else {
          fieldWidthTextBox.Text = "600";
          field_width = 600;
          tbLog.AppendText("\r\n----------------------------------------***");
          tbLog.AppendText("\r\nCannot exceed field Width more than" + (Screen.PrimaryScreen.Bounds.Width - 100)
+ " or less than 1");
          tbLog.AppendText("\r\n----------------------------------------***");
       }
    }
    this.Invalidate(); }
```

ComboBox1_SelectedIndexChanged Method:

This method is invoked when color is changed by updating the paint event.

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
  {
    colour = comboBox1.SelectedItem.ToString();
    this.Invalidate();
  }
```

DefaultBtn_Click Method:

This method is invoked when default button is clicked in the form. This changes the Height, width, color of the field back to default.

```
private void defaultBtn_Click(object sender, EventArgs e)
  {
    field_height = 600;
    field_width = 600;
    fieldHeightTextBox.Text = field_height.ToString();
    fieldWidthTextBox.Text = field_width.ToString();
    heightScrollBar.Value = field_height / 15;
    widthScrollBar1.Value = field_width / 15;
    colour = "Black";
    comboBox1.SelectedItem = "Black";
    this.Invalidate();
  }
```

Form1_Paint Method:

This method calls the paint event to display the field. This method draws the field using color, height and width of the field class variables

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
    {
      Console.WriteLine("fieldWidth " + field_width);
      if(colour == "Black")
        e.Graphics.FillRectangle(Brushes.Black, new Rectangle(500, 30, field_width, field_height));
      else if(colour == "Green")
        e.Graphics.FillRectangle(Brushes.LightGreen, new Rectangle(500, 30, field_width, field_height));
        else if (colour == "Turquoise")
        e.Graphics.FillRectangle(Brushes.Turquoise, new Rectangle(500, 30, field_width, field_height));
        else
          e.Graphics.FillRectangle(Brushes.White, new Rectangle(500, 30, field_width, field_height));  }
```

Button1_Click Method:

This method is invoked when pause button is clicked, stopping both the timers and enabling the logs.

```
private void button1_Click(object sender, EventArgs e)
    {// Pause button\
      if (pause == 0)
      {
        pause = 1;
        timer1.Stop();
        timer.Stop();
        tbLog.Enabled = true;
        logs.pausedTime();
      }
    }
```

BtnResume_Click Method:

This method is invoked when Resume button is clicked starting both the timers and disabling the logs.

```
private void btnResume_Click(object sender, EventArgs e) {
      logs.resumeTime();
      if (pause == 1) {
        timer.Start();
        targetsMove = new Thread(moveTargets);
        targetsMove.Start();
        pause = 0;
        tbLog.Enabled = false;} }
```

ResetBtn_Click Method:

This method is invoked when Resume button is clicked, which enables and disables a bunch of components as well as deleting all the drones and resetting the flags.

```
private void resetBtn_Click(object sender, EventArgs e) {
        timer.Stop();
        stopTargets();
        NumTargets.Enabled = true;
        NumUav.Enabled = true;
        FieldSettings.Enabled = true;
        startBtn.Enabled = true;
        button1.Enabled = false;
        btnResume.Enabled = false;
        defaultBtn.Enabled = true;
        resetBtn.Enabled = false;
        tbLog.AppendText("\r\n----------------------------------------------");
        tbLog.AppendText("\r\nReset at " + DateTime.Now.ToString(" HH:mm:ss"));
        pause = 1;
        tbLog.Enabled = true;
        button1.Enabled = false;
        btnResume.Enabled = false;
        for (int i = 0; i < targets.Length; i++)
            targets[i].deleteDrone();
        for (int i = 0; i < uavs.Length; i++)
            uavs[i].deleteDrone();
        Target.count = 0;
        Uav.count = 0;
        Uav.directionChangeFlag = 0;
        Uav.direction = 1;
    }
```

## Form1_Resize Method:

This method is called when the application window is resized to keep the field within sight.

```
private void Form1_Resize(object sender, EventArgs e) {
        if (this.Height < 740)
            this.Height = 740;
        else {
            if (heightScrollBar.Value * 15 > this.Height - 100)
                field_height = this.Height - 100;
            if (widthScrollBar1.Value * 15 > this.Width - 600)
                field_width = this.Width - 600;
            if (field_width <= 14)
                field_width = 15;
            this.Invalidate();
            fieldHeightTextBox.Text = field_height.ToString();
            fieldWidthTextBox.Text = field_width.ToString();
            heightScrollBar.Value = field_height/15;
            widthScrollBar1.Value = field_width / 15;
        }
        }
```

## Drones Class:

This is an abstract class used by both UAVs and Targets forcing them to use same hierarchies or standards.

```
abstract class Drones
    {
        protected int xCoordinates;
        protected int yCoordinates;
        protected PictureBox droneBox;
        protected Boolean inGameStatus;
        protected Boolean killedStatus;

        abstract public void move(int fieldHeight, int fieldWidth);
        abstract public int getXCoordinates();
        abstract public int getYCoordinates();
        abstract public PictureBox getDroneBox();
        abstract public void setKillStatus(bool p);
        abstract public Boolean getKillStatus();
        public  void deleteDrone()
        {
            droneBox.Visible = false;
            droneBox.Enabled = false;
        }
    }
```

## Target Class:

This class inherits the Drones abstract class.

Variables:

prevX and prevY is used to track X and Y coordinates for generating random new ones. Count is used to track number of selected targets.

```
class Target : Drones
    {
        int prevX;
        int prevY;
        public static Random rand;
        public static int count;
```

Default and Parameterized Constructor:

Default constructor is used to initialize random and count variables.

```
static Target() {
            rand = new Random();
            count = 0;
        }
```

Parameterized Constructor is invoked when new targets are generated initializing X and Y coordinates, prevX, prevY, simulator statuses, and invoking SetTarget method.

```
public Target(int fieldWidth, int fieldHeight, Form form1)
    {
        xCoordinates = rand.Next(500, fieldWidth - 15);
        yCoordinates = rand.Next(30, fieldHeight - 15);
        Console.WriteLine("x: " + xCoordinates + ", y: " + yCoordinates);
        prevX = xCoordinates;
        prevY = yCoordinates;
        inGameStatus = true;
        killedStatus = false;
        setTarget(form1);
        count++;
    }
```

SetTarget Method:

This method is used to generate the target on the field.

```
public void setTarget(Form form1)
    {

        droneBox = new PictureBox();

        Bitmap UavImg = new Bitmap(Properties.Resources.t1);

        droneBox.BackColor = Color.Transparent;
        droneBox.Image = (Image)UavImg;
        droneBox.Enabled = true;
        Size size = new Size(12, 14);
        droneBox.Size = size;
        droneBox.SizeMode = PictureBoxSizeMode.StretchImage;
        droneBox.Location = new System.Drawing.Point(xCoordinates, yCoordinates);

        form1.Controls.Add(droneBox);
    }
```

Move Method:

This method overrides move method from Drones abstract class. The code is only executed if the kill status of the target is not true. This method generates new x and y coordinates within specific radius using previous coordinates and random variable and moves the target to new location.

```
public override void move(int fieldHeight, int fieldWidth)
        {

            if (!killedStatus)
            {
                prevX = xCoordinates;
                prevY = yCoordinates;

                do
                {
                    xCoordinates = rand.Next(prevX - 15, prevX + 10);
                } while (xCoordinates >= fieldWidth - 15 || xCoordinates <= 505);

                do
                {
                    yCoordinates = rand.Next(prevY - 15, prevY + 10);
                } while (yCoordinates >= fieldHeight || yCoordinates <= 40);

                droneBox.Location = new Point(xCoordinates, yCoordinates);
            }
        }
```

## SetKillStatus Method:

This method overrides the setter method and initiates setKillStatus from Drones abstract class, while updating the kill status of the target.

```
public override void setKillStatus(Boolean status)
        {
            killedStatus = status;
            if (killedStatus)
                droneBox.Visible = false;

        }
```

## Getter Methods:

Following methods return the target information

```
public override Boolean getKillStatus()
        {
            return killedStatus;
        }
        public override int getXCoordinates()
        {
            return xCoordinates;
        }
        public override int getYCoordinates()
        {
            return yCoordinates;
        }
        public override PictureBox getDroneBox()
```

```
{
    return droneBox;
}
```

UAV Class:

This class inherits abstract class, Drones.

Variables:

Direction and directionChangeFlag variables are used to track the east/west direction of UAVs. Radar variable is used to track the movement through X axis and count variable is used to track the number of selected UAVs, while uavImg variable is the image of the UAV and it is used to flip the image on change of direction.

```
class Uav : Drones
    {
        public static int direction;
        public static int directionChangeFlag;
        private static int radar;
        public static int count;
        private Bitmap uavImg;
```

SetUav:

This method is used to generate UAVs on the field.

```
public void setUav(Form form1)
        {
            droneBox = new PictureBox();

            uavImg = new Bitmap(Properties.Resources.uav);

            droneBox.Image = (Image)uavImg;
            droneBox.BackColor = Color.Transparent;
            droneBox.Enabled = true;
            Size size = new Size(30, 20);
            droneBox.Size = size;
            droneBox.SizeMode = PictureBoxSizeMode.StretchImage;
            droneBox.Location = new System.Drawing.Point(xCoordinates, yCoordinates);

            form1.Controls.Add(droneBox);


        }
```

UpdateRadar Method:

This method is used to update the radar on each move. According to the radar value it is decided if the direction change is required.

```
static public void updateRadar(int fieldWidth)
    {
      if (direction == 1)
      {
        if (radar < fieldWidth + 5)
          radar++;
        else
        {
          radar--;
          direction = 0;
          directionChangeFlag = 1;
        }
      }
      else
      {
        if (radar >= 460)
          radar--;
        else
        {
          radar++;
          direction = 1;
          directionChangeFlag = 1;
        }

      }
```

Move Method:

This method overrides move method from abstract class, Drones. If the direction change flag is true, then direction is changed otherwise the X coordinates are updated with radar variable value.

```
public override void move(int fieldHeight, int fieldWidth)

{
   xCoordinates = radar;
  if (directionChangeFlag == 1)
  {
    if (yCoordinates > fieldHeight - 40)
    {
      uavImg.RotateFlip(RotateFlipType.Rotate180FlipY);
      yCoordinates = 30;
      direction = 1;
    }
    else
    {
      uavImg.RotateFlip(RotateFlipType.Rotate180FlipY);
      droneBox.Image = uavImg;
      yCoordinates += 15;
```

```
            }
        }
        droneBox.Location = new Point(xCoordinates, yCoordinates);
    }
```

Getter Methods:

Following methods return the UAV information

```
public override int getXCoordinates()
        {
            return xCoordinates;
        }
        public override int getYCoordinates()
        {
            return yCoordinates;
        }
        public override PictureBox getDroneBox()
        {
            return droneBox;
        }

        public override bool getKillStatus()
        {
            throw new NotImplementedException();
        }
        public override void setKillStatus(bool p)
        {
            throw new NotImplementedException();
        }
```

<u>LogBook Class:</u>

This class is used to update the onscreen log text box.

Variables:

Start and end variables are used to track the simulation start and end time, elapsedPauseTime variable is to track the pause time and the pause variable is used to track the time last pause was done.

```
class LogBook
 {
    DateTime start; // simulation start time
    DateTime end; // simulation end time
    TimeSpan elapsedPauseTime;
    DateTime pause;
    Boolean pauseFlag;
    TextBox tbLog;
```

Constructor:

Constructor initializes the textbox and pause flag.

```
public LogBook(TextBox textBox)
    {
       tbLog = textBox;
       pauseFlag = false;

    }
```

PausedTime Method:

This method is invoked when pause button is clicked, which enables it to log the exact time of the pause.

```
public void pausedTime()
    {
       pause = DateTime.Now;
       tbLog.AppendText("\r\n Simulation Paused at: " + pause.ToString("HH:mm:ss"));
       pauseFlag = true;
    }
```

StartTime Method:

This method is invoked when the start button is clicked logging the initial information about the simulation run.

```
public void startTime(int numOfTargets, int numOfUavs)
    {
       tbLog.AppendText("\r\n\r\n ==========================");
       tbLog.AppendText("\r\nNew Simulation");
       tbLog.AppendText("\r\n UAVs: " + numOfUavs.ToString());
       tbLog.AppendText("\r\n Targets:  " + numOfTargets.ToString());
       start = DateTime.Now;
       tbLog.AppendText("\r\n Simulation started at " + start.TimeOfDay + "\r\n\r\n");
    }
```

ResumeTime Method:

This method is invoked when resume button is clicked logging the elapsed time of pause.

```
public void resumeTime()
    {
       try
       {
          if (pauseFlag)
          {
             DateTime resumed = DateTime.Now;
             tbLog.AppendText("\r\n Simulation Resumed at " + resumed.ToString(" HH:mm:ss"));
             elapsedPauseTime = resumed - pause;
```

```
                pauseFlag = false;
            }
            else
                throw new System.ArgumentException("Warning on resume click: 'Simulation is already running!'");
        }
        catch (Exception e) {
            tbLog.AppendText(e.Message);
        }
    }
```

Finish Method:

This method is invoked when all the targets are killed to log the final simulation information in the logbook textbox.

```
public void finish(int numOfTargets, int numOfUavs, int fieldHeight, int fieldWidth, double maxSpeed)
    {
        end = DateTime.Now;
        tbLog.AppendText("\r\n ----------------------------------------");
        tbLog.AppendText("\r\n All Targets are killed ");
        tbLog.AppendText("\r\n No. of UAVs: " + numOfUavs);
        tbLog.AppendText("\r\n No. of Targets: " + numOfTargets);
        tbLog.AppendText("\r\n Field Height: " + fieldHeight);
        tbLog.AppendText("\r\n Field Width: " + fieldWidth);
        tbLog.AppendText("\r\n UAVs Max Speed: " + maxSpeed);

        tbLog.AppendText("\r\n Simulation ended at " + end.TimeOfDay);
        TimeSpan diff = end - start - elapsedPauseTime;
        tbLog.AppendText("\r\n\r\n Simulation Time excluding paused time : \r\n " + diff);
        diff = end - start;
        tbLog.AppendText("\r\n\r\n Simulation Time including paused time : \r\n " + diff);

    }
```

The simulator is built using classes, methods and many variables, which are initiated as soon as the simulator application starts, also indulging the end user in selecting and choosing various options of his or hers choice. The next section which assesses the future of UAV technology, points to studies that will help expand and upgrade the simulator in the coming years.

# 5. FUTURE STATE

The purpose of this section is to give an understanding of what the simulator will look like in the future and what can be achieved in the coming research. There are different features that can be added to the simulator using the linear programming approach taken from Dr. Kendall Nygard's paper.

Future integrations that will enhance the simulators functions will be:

- Dynamic UAVs
- Communication

These features would make the simulator completely hands free. The user or controller would have to enter the destination of the target, number of targets, and the number of UAVs to engage in the strike. Both features are hard to integrate in a UAV, as it would require an additional part that must be implanted. However, once integrated, it would make UAVs fully automated.

To make the UAVs fully automated and have multiple UAVs strike a specific target (whether it be static or moving), UAVs need to have a communication feature added into their system or program, which allows data sharing between all UAVs. This not only provides important information from one UAV to another, but also information about the location of a target that might be in range for one UAV, but not for another.

This would also allow communication about impending obstacles that might be in the way of the targets. There was research conducted by Niels Damgaard Hansen [17], where they extend colored Petri Nets with channels to create synchronous communication. Not only does the use of colored Petri Nets increase the chances of creating a working model for communication between the UAVs, it also helps in assuming the properties of Petri Nets with the channels form other Petri Nets.

Using the linear programing functionality from Kendall Nygard et al. [1, 2, 3], study, we can add the communication agent to the simulator; as already explained, the linear programming is useful in adding or removing features from the simulator easily without having to change the whole program or system. With the help of the communication agent embedded in the simulator, we can show the correspondence between multiple UAVs. Using this device, UAVs would be able to transmit data and information to the closest UAV (or UAV within range) and pass along the information from A to B to C.

Addition of the communication agent in the simulator is the first step in making the UAVs fully autonomous. The benefits of this feature can also be used in search and rescue expeditions as well.

The other component that can be added to the UAVs is radar, or implementing a sensor. This would enable UAVs to perceive obstacles that might be in their paths and share this information. Adding a radar or sensor can be costly though. These systems are expensive and heavy to be fitted on UAVs. Even creating a connection to GPS satellites through UAVs can be costly. In addition, UAVs also carry the risk of being destroyed or damaged, adding to overall cost.

One study, conducted by Sanjiv Singh[4], brings attention to how the addition of these features can be costly and useless for smaller and lighter UAVs. To overcome this limitation, an algorithm was developed that uses a camera and several software programs to detect obstacles. This reduces the cost of embedding a large radar or sensor into the UAVs that in turn, would slow them down and use much of the UAVs power.

Using the same strategy as proposed by Sanjiv Singh [4]**,** to use a camera instead of radar or large sensors, could benefit the UAVs. Embedded UAV cameras detect obstacles in their paths and change directions, attack a single target, or several targets. This also keeps the weight of the UAVs in proportion with the power needed to operate them.

The use of these features in UAVs, as well as adding these agents to the simulator, improves the chance of having a better striking graph and brings UAVs closer to being fully automated. These added components can also help test UAVs in uncontrolled environments or guide UAVs to move from certain points or locations without any help from a human controller.

Using these two upgrades, UAVs can also achieve a multipoint attack simulation. They will be able to communicate the distance of a target, avoid obstacles, and attack at different angles on the same target or multiple targets, which would achieve a better strike graph.

The integration of the Petri Nets in the future in this simulator and the UAVs can enhance the system a lot. The communication and the dynamic movements of the UAVs using the Petri Nets and Emergent behavior can move us closer to our goals for the system and the UAVs. These improvements can not only get us closer to UAVs which are capable of performing a successful search and rescue mission or even get us closer to the attack rate of a complete hundred percent as multiple UAVs attacking a randomly moving target can get us closer to our desired results.

Ultimately, these upgrades can be useful for the simulator; they can assist militaries around the world to combat terrorism, reduce causalities, and be used for carrying out search and rescue missions.

# 6. CONCLUSION

The main resolution of this paper is to help improve Unmanned Aerial Vehicles as they increase in popularity for military and civilian use, potentially saving more lives. Using this paper and many other related papers, we can come up with fully autonomous UAVs that can be used for almost any tasks that are assigned to them.

The simulator defined in this paper, combined with future upgrades and research will give it an edge over many studies that are currently being conducted. The simulation will help visualize the concept of multipoint attack by UAVs on several targets and demonstrates how they communicate with one another. This simulator would not only benefit militaries, but other civilian projects too, such as search and rescue, exploration and agriculture.

In conclusion, this simulator is a stepping stone, a foundation to help design better programs, which can adapt to the situations and achieve the desired results without depending on humans to guide UAVs to the targets.

## 7. REFERENCES

[1] Xu, D., Borse, P., Grigsby and K., Nygard, “A petri net based software architecture for UAV simulation,” presented at the proceedings of Software Engineering Research and Practice (SERP04), 2004.

[2] C. Lua, K. Altenburg, and K. Nygard. “Synchronized Multi-Point Attack by Autonomous Reactive Vehicles with Simple Local Communication.,” presented at the proceedings of the IEEE Swarm Intelligence Symposium, Indianapolis, Indiana, 2003.

[3] N. Huff, A. Kamel, and K. Nygard, “An Agent Based Framework for Modeling UAVs.,” presented at the proceedings of the 16th International Conference on Computer Applications in Industry and Engineering (CAINE03), 2003.

[4] Geoffrey Hollinger, Sanjiv Singh, Joseph Djugash, and Athanasios Kehagias, “Efficient Multi-Robot Search for a Moving Target,” The International Journal of Robotics Research, Vol. 28, No. 2, pp. 201-219 February, 2009.

[5] K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Springer Verlag, 1992.

[6] Y. Altshuler, V. Yanovsky, I. A. Wagner, and A. M. Bruckstein, “Efficient cooperative search of smart targets using UAV Swarms,” Robotica, vol. 26, no. 4, pp. 551–557, 2008.

[7] Y. Altshuler, V. Yanovsky, I. Wagner, and A. Bruckstein, “The cooperative hunters - efficient cooperative search for smart targets using UAV swarms.” presented at the proceedings of the Second International Conference on Informatics in Control, Automation and Robotics (ICINCO), 2005.

[8] S. Christensen and N.D. Hansen, “Coloured Petri Nets Extended with Channels for Synchronous Communication.” presented at the proceedings of 15th International Conference on

the Application and Theory of Petri Nets, Lecture Notes in Computer Science 815, pp 159–178, Zaragoza, Springer-Verlag (1994).

[9] Qiang Liu, Ming He, Daqin Xu, Ning Ding, and Yong Wang, “A Mechanism for Recognizing and Suppressing the Emergent Behavior of UAV Swarm,” *Mathematical Problems in Engineering*, vol. 2018, Article ID 6734923, 14 pages, 2018. https://doi.org/10.1155/2018/6734923.

[10] Delin Luo, Wenlong Xu, Shunxiang Wu and Youping Ma, "UAV formation flight control and formation switch strategy," presented at the proceedings of 8th International Conference on Computer Science & Education, Colombo, pp. 264-269, 2013.

[11] Bandala, Argel & Dadios, Elmer & Rhay P. Vicerra, Ryan & Gan Lim, Laurence. “Swarming Algorithm for Unmanned Aerial Vehicle (UAV) Quadrotors: Swarm Behavior for Aggregation, Foraging, Formation, and Tracking.” Journal of Advanced Computational Intelligence and Intelligent Informatics. vol.18, pp.745-751, 2014.

[12] Shweta Singh, “Detection of Emergent Behaviors In System Of Dynamical Systems Using Similitude Theory.” presented at the Winter Simulation Conference, 2017.

[13] Andrea Bertolaso, Masoume M. Raeissi, Alessandro Farinelli and Riccardo Muradore. “Cooperative UAV-UGV modeled by Petri Net Plans specification.” presented at the proceedings of Bertolaso 2016 Cooperative UM. 2016