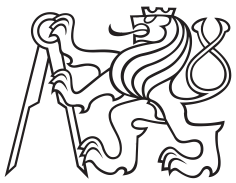
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Critical Infrastructure Protection Against Attacks Using Drones

Jan Kúdelka

May 2018

Supervisor: Ing. Milan Rollo, Ph.D.

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kúdelka Jan** Personal ID number: **456945**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Critical Infrastructure Protection Against Attacks Using Drones

Bachelor's thesis title in Czech:

Ochrana kritických infrastruktur před napadením s využitím bezpilotních prostředků

Guidelines:

1. Study the problem of small unmanned aerial systems (UAS) misuse for attacks on critical infrastructure.
2. Survey existing sensors for detection of small UAS.
3. Design and implement simplified models of detection sensors.
4. Design and implement software architecture for simulation of attacks and their detection.
5. Experimentally evaluate the ability of counter-UAS systems to detect different types of attacks based on combination of various sensors.

Bibliography / sources:

[1] Giorgia Sinibaldi, Luca Marino.: Experimental analysis on the noise of propellers for small UAV, In Applied Acoustics, Volume 74, Issue 1, pp. 79-88, 2013.
[2] Gabriel C. Birch, John C. Griffin, and Matthew K. Erdman.: UAS Detection, Classification, and Neutralization: Market Survey 2015, Sandia National Laboratories, 2015.
[3] Minas Benyamin and Geoffrey H Goldman: Acoustic Detection and Tracking of a Class I UAS with a Small Tetrahedral Microphone Array, Army Research Laboratory, 2014.

Name and workplace of bachelor's thesis supervisor:

Ing. Milan Rollo, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **09.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Milan Rollo, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt

Student's signature

/ Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 20 May 2018

.....

Abstrakt / Abstract

V posledních letech se zvýšila míra používání dronů a také jejich přístupnost veřejnosti. Tudíž se drony staly problémem pro celou řadu lidí, jelikož mohou být jednoduše využity k ilegálním aktivitám. Tyto aktivity zahrnují pašování, sabotáž, špionáž a používání dronů v místech, kde je veliká pravděpodobnost nehod. Kvůli tomu je nyní zvětšující se potřeba drony detekovat a bránit se před nimi. V této práci bude navrženo simulační prostředí, které bude použito pro simulaci vícesenzorového systému pro detekci dronů ve snaze pokrýt navrženou oblast senzory, aby došlo k detektci všech dronů, které skrz ni letí.

Klíčová slova: UAV, UAS, Drony, Detekce Dronů, Protekce Infrastruktury, Simulace, Prostředí

In recent years, the use of drones has increased in popularity and so has their availability to the public. Consequently, drones have become a problem for many people, as they can be easily used for certain illegal activities. These activities include smuggling, sabotage, espionage, or usage in places where accidents are highly probable. As a result, there is an increasing need for detection of drones and protection from them. In this paper, a simulation environment will be designed and used to simulate a multi-sensor system for drone detection in an attempt to fully cover an area with sensors, to detect any UAV passing through it.

Keywords: UAV, UAS, Drones, Drone detection, Infrastructure Protection, Simulation, Environment

Contents /

1 Introduction 1
1.1 Drone threats in detail 1
2 State of the Art 3
2.1 Drone types 3
2.1.1 Flying technology 3
2.1.2 Size and weight 3
2.1.3 Power source 3
2.3 Radars 5
2.4 Acoustic sensors 6
2.5 Cameras 7
2.6 Current solutions 8
2.6.1 Liteye 8
2.6.2 Anti-Drone 9
2.6.3 Droneshield 9
3 System Design 10
3.1 The simulation world representation 10
3.2 Sensor models 10
3.3 Visualisation 10
3.4 Detectable objects 11
3.5 Design summary 11
4 Essential algorithms, structures and principles 12
4.1 Ray tracing 12
4.2 KD-Tree data structure 12
4.2.1 Structure 12
4.2.2 Surface Area Heuristic 13
4.2.3 Plane sweeping 13
4.2.5 KD-Tree traversal 15
4.3 Intersection algorithms 16
4.3.1 Ray-triangle intersection 16
4.3.2 Ray-box intersection 17
4.3.3 Ray-sphere intersection 19
5 Sensor modelling with ray-tracing 20
5.1 Camera 20
5.1.1 Model Matrix 20
5.1.2 View Matrix 21
5.1.3 Projection Matrix 21
5.1.4 Projection Space to Image Space 21
5.1.5 Image to world space transformation 22
5.2 Acoustic sensor modelling 22
5.2.1 Sound absorption while travelling 22
5.2.2 Surface-related sound absorption 22
5.2.3 Sound reflection and refraction. Energy distribution functions. 23
6 Implementation 24
6.1 Architecture 24
6.2 Implemented classes and improvements 24
6.2.1 KD-Tree 25
6.2.2 CameraSensor, CameraObject and CameraVisual 25
6.2.3 AcousticSensor 26
6.2.4 OBJLoader 27
6.2.5 Ray and AcousticRay 27
6.2.6 AABB 27
6.2.7 Triangle 27
6.2.8 IntersectionHandler 27
6.2.9 Drone 28
6.2.10 Settings 28
6.2.11 ReflectionGenerator 28
7 Experiments 29
7.1 Experiment 1 - Closed space scenario 29
7.1.1 Results 29
7.2 Experiment 2 - Camera max distance scenario 31
7.2.1 Result 31
7.3 Experiment 3 - Acoustic sensor max distance scenario 32
7.3.1 Results 33
7.4 Experiment 4 - Multiple-sensor scenario 34
7.4.1 Results 35
7.5 Designing a sensor area coverage model for real infrastructure 35
7.6 Generating custom scenarios 35
8 Conclusion and Future work 36
8.1 Future work 36
8.2 Conclusion 36
References 38

Chapter 1
Introduction

In recent years, drones have become available for purchase worldwide. In this paper, the term drone is interchangeable with the term LSS-UAS(low, small and slow unmanned aerial systems). There are several reasons for using drones for malicious purposes [1].

- Simple attack preparation
- Difficulty to detect
- Can carry cameras/weapons
- Can be smuggled into a protected area and then used to carry out an object or data

The US Department of Homeland Security categorises drone-related threats as [2]:

- Weaponized or Smuggling Payloads
- Prohibited Surveillance and Reconnaissance
- Intellectual Property Theft
- Intentional Disruption or Harassment

It can be seen that drones are a very versatile tool for malevolent activity. It is therefore needed to find a sufficient means of detection to neutralise these threats. Drones can have various aspects, that need to be faced to correctly detect them. They can be very small hence very hard to detect. They can use camouflage to hide from cameras. They are not always very loud therefore they are hard to detect acoustically - some drones can ascend into high altitudes and then silently glide towards their destination. Some drones are made of materials that isolate heat and have a very small radar cross-section. These challenges can be overcome only by using a very well designed and thoroughly examined system.

1.1 Drone threats in detail

The weaponized payloads in the first category of threats imply the use of drones to carry guns or explosives with added triggering mechanisms to make them usable for hurting people or damaging infrastructure. There have been many occasions at which terrorists, armies, civilians or defence companies put guns on drones [5–7]. Smuggling payloads in the same category mean drones can be used for transportation of drugs or other items. This is becoming very common all across the world [8–9].

Prohibited surveillance means drones have been used for surveilling infrasctructure or people's activities. This could include industrial espionage, terrorists using drones to gather intelligence about military activity or simple spying on people by civilians for various purposes. Many countries have already taken steps to regulate drone usage, because of surveillance. A list of countries and restrictions, which is updated by the public, can be found at [10]. Surveillance drones can also be used to steal intellectual property for example by getting information about company secrets.

Drones are also being used for intentional disruption or harassment. This means operators are using drones to follow people around or intentionally fly over other person's property.

Some of the biggest threats of drone usage are also the unintentional ones. Some drone operators use drones in irresponsible ways or in restricted areas without their knowledge. These areas include airports, military infrastructure and areas susceptible to damage such as areas around monuments and old buildings.

This thesis is organised as follows. In Chapter 2, the state of the art of drone detection is studied as well as drone characteristics that are important for their detection. In Chapter 3, the architectural structure of the simulation system will be discussed. Chapter 4 talks about essential algorithms and data structures used in the simulation environment. Chapter 5 follows up on Chapter 4 and specifies the use of the ray tracing algorithm in sensor modelling. The implementation and optimalisation of the system is discussed in Chapter 6. It is then tested in Chapter 7 where drone detectability by acoustic sensors and cameras is also examined. The thesis is finally concluded in Chapter 8.

Chapter 2
State of the Art

This chapter provides a summary of the research done on the topic of drones, drone threats, current options for drone detection and already applied solutions to the detection problem.

2.1 Drone types

It is important to understand the different characteristics of drones in order to properly deal with them when detecting drones [4].

2.1.1 Flying technology

The first characteristic of a drone is the technology it is using to stay in flight. There are only two notable categories; rotary-wing and fixed-wing based drones.

Drones in the first category use rotary wings or 'rotors' to generate lift. Most drones use 3 or more of these rotors. Heavier drones tend to either have bigger wings or to have more of them as they require bigger upthrust to stay in the air. This flying technology is the most used one, especially by the public. These drones can be easily stabilized and do not need to keep moving in order to stay in the air. Drones with fixed-wings use forward airspeed to generate lift force. The source of their speed is the conversion from potential energy to kinetic energy by using gliders sometimes combined with a jet engine.

If a drone is in the first category, there is sure to be a certain minimal constant noise level from the rotors, because if the rotors are turned off the drone is not going to be able to stay in the air. The second category implies the drone's ability to glide and stay silent for some time. Gliding drones, on the other hand, have wings which make them larger and easier to spot by certain sensors.

Examples of drones with these technologies can be seen in Figure 2.1. and Figure 2.2.

2.1.2 Size and weight

Another important characteristic is the size and weight of a drone. It is generally easier to detect larger and heavier objects as they have larger radar cross section, are easier to detect using cameras and produce more noise and heat. This thesis focuses on drones that are small and light and are therefore a challenge to detect. In the sensor section it will be discussed how the size and weight of the drone affect their detectability.

2.1.3 Power source

The last main characteristic of drones mentioned in [4] is their power source. These are mainly battery cells, fuel cells or solar cells. Solar cells are not used very commonly. Their use would require the drone to be larger but also lighter. Battery cells are used by most small drones, produce little heat and therefore do not make it easier to detect the

drone. Fuel cells are heavier than battery cells and are therefore used on heavier drones or ones with fixed-wing technology. Most of these cells should also produce detectable heat.

Figure 2.1. Rotor-based drone example: CyPhy - Rotor Drone Magazine [1]

Figure 2.2. Fixed-wing drone example: Sentera Phoenix 2 - PrecisionAg [2]

2.1 Means of detection

Different sensor types can be used to detect drones [3]:

- Acoustic sensors
- Passive cameras (UV and visible light cameras)

[1] Reid, John. Drone with rotor based technology. Rotor drone magazine, May 2015, www.rotordronemag.com/wp-content/uploads/2015/05/cyphy-level-one-drone-640x300.jpg.
[2] Hopkins, Matt. Drone with fixed-wing technology. PrecisionAg, June 2016, www.precisionag.com/systems-management/sentera-debuts-fixed-wing-precision-agriculture-focused-drone.

- Passive thermal cameras
- RF emission sensors
- Radars
- Magnetic detection systems

For this thesis 3 sensor types have been selected - radars, acoustic sensors and cameras operating in the visible spectrum. A system consisting of these 3 types should be able to detect many different types of drones. The reasons behind this decision are:

1. Most LSS drones do not produce enough heat for the use of thermal cameras. [3]
2. RF emission sensors are not sufficient as only remotely controlled drones emit RF energy.
3. Most LSS drones do not use enough metal parts to be detected at an adequate range by magnetic detection systems. [3]
4. It is assumed that drones that are designed to specifically avoid cameras, acoustic sensors or radars will be detected by other sensors from this group.

2.3 Radars

Radar is a system for detecting the presence, direction, distance and speed of objects. The radar works by first radiating electromagnetic energy from an antenna. Some of this radiated energy hits the object (target) and is reflected. A fraction of this reflected energy is then picked up by the receiving antenna. After amplifying the received signal and processing it, it is decided whether the object is present [11]. An example of a radar can be seen in Figure 2.3.

Radars are active sensors that can operate during both day and night. The main challenge with a radar is the small radar cross-section of small drones [20]. In order to detect an object by a radar, the reflected signal from the object must be strong enough. The signal needs to be stronger the stronger the background noise is. Therefore SNR (Signal to noise ratio) is an adequate parameter that determines if the simulated radar is detecting a drone. The SNR of the signal needs to be higher than 15 dB for a radar to detect a drone [12]. The equation for calculating SNR is:

$$SNR = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 r^4 k T B_n}$$

where

P_t	transmitter power
G_t	gain of the transmitting antenna
G_r	gain of the receiving antenna
λ	wavelength of radio wave
σ	radar cross section of an object
k	Boltzmann's constant
T	receiver noise equivalent temperature
B_n	processing noise bandwidth
r	distance from the transmitter to the target

From the required SNR a maximum distance of detection r_{max} can be calculated. Another condition for detecting is that the drone's velocity must be lower than a certain velocity v_{max}. This velocity can be calculated as [13]:

$$v_{max} = \frac{c}{2B\tau}$$

where

c	speed of light
τ	radar's integration time
B	radar's bandwidth

Figure 2.3. Radar example: SQUIRE - Thales [1]

2.4 Acoustic sensors

Acoustic sensors, in general, are sensors that detect acoustic waves (sound) emitted by an object. The main source of these waves for drones are propellers and engines. Usually, if a drone is heavier it also makes more noise when flying, because the propellers need to move faster or be larger. Acoustic sensors are passive sensors and like radars can operate during both day and night. An example of an acoustic sensor can be seen in Figure 2.4.

The main issue with acoustic sensors is the background noise. Acoustic sensors have extreme trouble detecting anything in noisy areas (mainly cities) [19]. Acoustic sensors are very dependent on the implementation of a signal analyzer. It has been shown that

[1] SQUIRE radar by THALES. Thales, https://www.thalesgroup.com/sites/default/files/squire/images/image-3.jpg

good implementations show up to 99.5% chance of detection with only 3% false alarm probability [21].

Figure 2.4. Acoustic sensor example: Orelia - DroneBouncer [1]

2.5 Cameras

Camera-based sensors consist of real-time recording and image analysis. Cameras can either be completely static or mobile, such as PTZ (pan, tilt, zoom) cameras, which have means of tracking the detected objects. Such a camera can be seen in Figure 2.5. With the improvements to computer vision [22] and camera image resolution [23] in recent years, cameras have become a viable means of detecting and classifying objects. Their main downsides are their dependency on light conditions and false positives when

[1] Orelia acoustic radar. Drone bouncer, http://dronebouncer.com/uploads/images/Orelia/ORELIA-0.jpg

detecting [24]. It has been concluded that combining cameras with other sensors can drastically increase their precision rate [24].

Figure 2.5. Camera example: M7 Camera - Tactical Imaging Systems [1]

2.6 Current solutions

Many companies have come up with solutions to the drone threat with their own detection systems.

2.6.1 Liteye

The company Liteye is using a multisensor system for detecting drones consisting of a radar and a video tracker. Its specifications are [14]:

Blighter A400 Series Air Security Radar

- Detection range: 8 km
- Minimum target size (RCS): 0.01 m^2
- Frequency band: Ku-band
- Radar type: E-scan Frequency Modulated Continuous Wave (FMCW) Doppler Surveillance Radar
- Power output: 4 Watt
- Azimuth coverage: 180° (standard) or 360° (optional)
- Elevation coverage: 10° (M10S antennas) or 20° (W20S antennas)
- Elevation adjustment: +/-40° using optional Blighter Radar Tilting System (BRTS)

Viper Dynamic Positioner:

- Azimuth: Continuous
- Elevation: -50° to +60°

[1] M7 Pan Tilt zoom LWIR Uncooled / MWIR Cooled Long Range Thermal PTZ FLIR Imaging camera for mobile EOIR Defense, UAV/Drone Detection, security & Surveillance. Tactical Imaging Systems, http://tacticalimaging.com/long-range-flir-thermal-ptz-pan-tilt-zoom-radar-night-vision-stabilized-gimbal-for-security-surveillance/

- Max speed: 60° per second

Piranha 46 HR Camera:

- Type: Colour HD 2.3 MP
- Optical zoom: x30
- Digital zoom: x12
- Focus: Auto

Thermal Camera:

- Type: Gen 3 Cooled
- Resolution: 640 x 512 pixel
- Wavelength: 3 to 5 μm
- Zoom: 24° to 1.8° FOV

EO Video Tracker:

- Type: Vision4ce digital video tracker and detector

2.6.2 Anti-Drone

The company Anti-Drone offers many different solutions, with multiple radars, cameras and acoustic sensors. One sensor in each category will be described here:

SpotterRF C550/C550EXT [15]

- Maximum Detection Range: 1500 m Vehicle, 1050 m Walker
- Coverage Area: 81.3 ha
- Effective FoV: Vertical: 20°, Horizontal: 90°
- Scan Rate: Up to 7 times per second
- Transmit Frequency: 10-10.6 GHz
- System Power: 8-13 Watts

Long-range acoustic sensor DroneShield v2 [16]

- Maximum Detection Range: 500 m Suburban, 1000 m Rural

Infrared and Video Surveillance Systems

2.6.3 Droneshield

The company Droneshield only shares its sensors' maximum detection ranges for their drone detection system Sentry, which are [17]:

- RadarOne: 1.5 km
- WideAlert: 100 m
- RFOne: 1 km
- DroneHeat/DroneOpt: 600 m for small drones and 2 km for large drones
- DroneCannon Engagement Range: 1 km

There are many other companies with their own solutions to the detection problem, but only those which have comprehensive and accessible data sheets of their sensors have been described. The ones which were not included are for example SRC, DeTect, Torrey Pines Logic, IEC Systems, Dedrone, Orelia and CellAntenna.

Chapter 3
System Design

One of the main goals of this work is to design a simulation environment. Several demands have been set when designing this environment:

- Precision
- Easy setup
- Real-time simulation
- Visualisation
- Swappable models of real sensors
- Integrability into the AgentFly framework

A system design is proposed in this chapter, which was devised to meet the above listed criteria.

3.1 The simulation world representation

For the world representation, something precise and easy to visualise needed to be chosen. The AgentFly framework already includes multiple .obj and .mtl files of closed and open spaces therefore triangle meshes derived from these files have been chosen. Triangle meshes provide an accurate world representation that is well fit for visualisation.

3.2 Sensor models

It has been decided to only model acoustic sensors and cameras at this stage. Sensors should be easily swappable and fast to set up therefore they are represented as a position in space with additional attributes, such as the FOV, direction and up vector for the camera. This allows the user to set up or swap out a real sensor by looking up its basic specifications and inserting them into the framework. The modelling of the sensors will be described in detail in the next chapters. All the sensors will use the triangle mesh to detect objects in it.

3.3 Visualisation

One of the criteria for the system is the possibility of visualisation of the sensors in it. Camera detection works by visualising the environment therefore it already meets this requirement. As far as the acoustic sensors are concerned, the AgentFly framework already has a powerful tool to visualise vectors and points in an environment, so the acoustic rays and their reflections in the environment should be visualised quite easily as well.

3.4 Detectable objects

The objects of the environemnt that are to be detected in it should also consist of a triangle mesh. This makes the detection by the camera very precise, as it can easily tell which triangle is the closest to it. For acoustic detection, a sound level attribute should be also added to the object.

3.5 Design summary

The simulation environment therefore consists of the main triangle mesh which represents the world, sensors which generate data about detection, visualisation frameworks that the data is sent to, and objects with their own triangle meshes and other attributes that are necessary. The environment is connected using principles discussed in Chapter 4 and the sensor modelling is discussed in Chapter 5.

Chapter 4
Essential algorithms, structures and principles

After designing a detection system and the environment, essential algorithms need to be examined. These algorithms include work with rays for visibility determination, building and tracing of related data structures and intersections with geometrical primitives.

4.1 Ray tracing

To successfully tell if an object is detected by a sensor, it is necessary to sufficiently test an object's visibility to the sensor. The ray tracing algorithm has been chosen for this in this paper.

As the name suggests, the ray tracing algorithm traces rays along their paths looking for the nearest intersection with an object. This is mostly used for high-quality rendering of images, as the rays can also be traced further with reflections creating very real-looking images [25]. In its basic form, ray tracing can be used to precisely tell if an object is visible from a sensor. All that needs to be done is to determine if the object is the closest object to the sensor (camera) or the point of reflection (acoustic sensor). This can be done as [27]:

```
for each object in scene do
    if ray intersects object and intersection is nearest so far then
        record intersection distance and object
return nearest object
```

This approach is very simple and easy to implement, it is ,however, very computationally demanding and therefore extremely slow [26]. Most objects will never be intersected by the algorithm, yet the intersection will still be computed. In more complex scenes, where there is a large number of objects, a data structure designed to cut down on the number of computations needs to be implemented. The data structure used in this environment is the KD-tree.

4.2 KD-Tree data structure

Many data structures that are fit for ray tracing have been introduced in the recent years, but KD-Trees have become one of the most used ones [28].

4.2.1 Structure

A KD-Tree is a data structure based on partitioning data and recursively dividing it over a splitting hyperplane. Each division splits the tree into two subtrees. Data is sorted according to the dimension of the split. Each object with a lower value on the splitting dimension's axis than the splitting plane becomes a part of the left subtree, the objects

with a higher value become parts of the right subtree [29]. In this implementation, the data that is being split consists of geometrical primitives - triangles. Sometimes, a triangle is split into two parts by the chosen splitting plane. This triangle will be added to both the left and right subtree.

In this work, the fast tracing of the KD-Tree is preferred over fast building. The tree will be built only once before the simulation begins therefore it is unnecessary for the building algorithm to be optimal. The tree will be implemented according to [28], which suggests an effective way of building KD-Trees. The main issue with KD-Tree building is the choice of the best splitting plane for data partitioning. Several heuristics can be chosen for this, but a surface area heuristic seems to be the best approach for splitting a space of geometrical primitives [30] .

4.2.2 Surface Area Heuristic

The surface area heuristic, or SAH, estimates the cost of a split by considering the probability of a ray passing through the two sub-voxels A and B, created by splitting a voxel by the plane, and the number of triangles that are in the newly created sub-voxels. The cost is calculated as:

$$C(A,B) = K_T + p_{[V_A|V]} \sum_{i=1}^{N_A} K_I(a_i) + p_{[V_B|V]} \sum_{i=1}^{N_B} K_I(b_i)$$

where

K_T	cost of traversal
K_I	cost of calculating intersection
$p_{[V_{sub}\|V]}$	The probability of hitting a sub-voxel V_{sub} if the voxel V is hit

The constants K_T and K_I vary depending on the scene.

The probability of hitting a sub-voxel, when the parent voxel is hit is assumed to be:

$$p_{[V_{sub}|V]} = \frac{SurfaceArea(V_{sub})}{SurfaceArea(V)}$$

With the heuristic set up in this way, it is also easy to determine, when to stop splitting. This happens when the cost of the split is higher than simply intersecting all the triangles:

$$Term = C_{best} > K_I|T|$$

A last modification to the cost calculation is proposed in [28]. By artificially further reducing the cost of splits with empty voxels we can get rid of empty space. The cost of these splits will, therefore, be multiplied by 0.8.

4.2.3 Plane sweeping

For better performance, a plane sweeping algorithm is implemented. The algorithm uses the fact that SAH only changes when the value is increased enough, so that a triangle begins or ends. This means there is a finite number of potential splits. These changes are called event, and by sorting these events and only going from event to event it is possible to sweep through the possible planes and count their SAH values.

Afterwards, the plane with the lowest cost is selected. An example of event distribution can be seen in Figure 4.1.

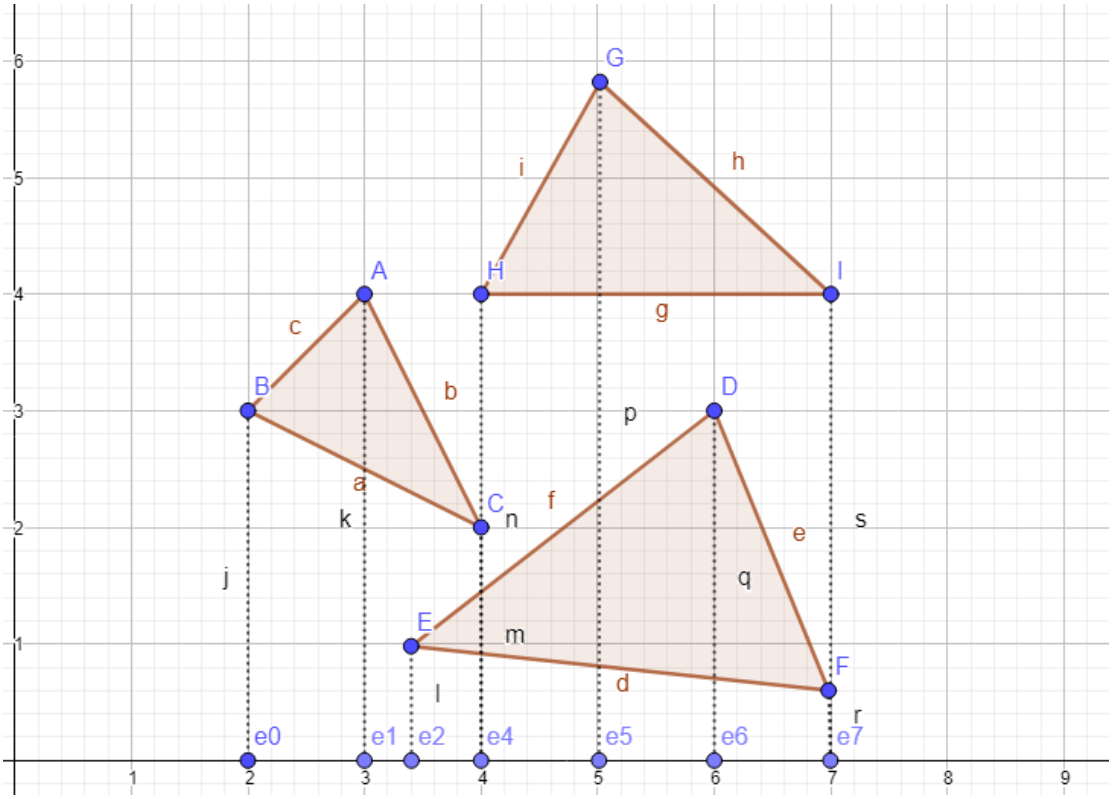


Figure 4.1. 2D Plane sweeping example on the x-axis. Points e0-e7 represent events.

4.2.3 Building algorithm

With the means of finding the best split set up, the KD-Tree can be built from the triangles that the environment consists of. The basic building algorithm uses recursion to build the tree, finding best splits and splitting until termination term becomes true. The algorithm is implemented as:

```
recBuildKDTree(Triangle[] triangles, AABB vox, int depth)
        begin
                index = treeList.size();
                p = findPlaneWithSweep(triangles, vox);
                term = terminate(p,triangles);
                if term then
                        leafIndex = leafList.size();
                        treeList.add(leafIndex);
                        leafList.add(triangles);
                else
                        treeList.add(Long.valueOf(p.coordinate << 29));
                        AABB vl, vr = split(vox,p)
                        Triangle[] tl,tr
                        tl = trianglesInVoxel(triangles, vl, p);
                        tr = trianglesInVoxel(triangles, vr, p);
                        recBuildKDTree(tl, vl, ++depth);
                        int rightIndex = recBuildKDTree(tr, vr, depth);
                        treeList.set(index, dataValue(rightIndex,p);
                end
```

```
                return index;
        end
```

AABB, in this algorithm, means the axis-aligned bounding box - the box that is getting split by the splitting plane. As far as the triangles that lie right on the splitting plane are concerned, they are first added to one sub-voxel then the other and the lower cost solution is taken as the best.

4.2.5 KD-Tree traversal

In order to use the KD-Tree more efficiently, a strong traversal algorithm must also be implemented. M. Hapala and V. Havran in their Review: KD-Tree Traversal Algorithm for Ray Tracing [31] suggest the use of three different methods used in the traversal algorithm: sequential, recursive and using neighbour-links. It has been decided to use the sequential traversal algorithm from their work, as it is well fit for parallel implementations, which might be necessary if a substantial amount of sensors is used at once in the simulation. The algorithm is described as follows:

```
Locate Leaf (node, point)
begin
         current node = node;
         if point lies outside node's AABB then
                 return no leaf exists;
         end
         while current node is not a leaf do
                 if point is to the left then
                         current node = cn's left child;
                 else
                         current node = cn's right child;
                 end
         end
         return current node;
end
Tree Traversal (ray)
begin
         (entry distance, exit distance) =
          intersect ray with root's AABB;
          if ray does not intersect AABB then
                 return no object intersected;
         end
         if ray has origin in AABB then
                 point = ray origin;
         else
                 point = ray origin + ray direction *
                 (entry distance + eps);
         end
         /* this will locate first leaf */
         current node = Locate Leaf ( tree root node, point);
         while current node is leaf do
                 /* current node is a leaf while
                 point is inside tree root node's AABB */
                 (entry distance, exit distance) =
                 intersect ray with current node's AABB;
                 if current node is not empty leaf then
```

```
                        intersect ray with each object;
                        if any intersection exists inside the leaf then
                                return closest object to the ray origin;
                        end
                end
                /* point just a bit outside the current node */
                point = ray origin + ray direction *
                (exit distance + eps);
                current node = Locate Leaf (
                tree root node, point);
        end
        return no object intersected;
end
```

4.3 Intersection algorithms

It is compulsory for ray tracing to implement several intersection algorithms with geometrical primitives. These include ray-triangle intersection and ray-box intersection which are used for tree traversal and ray-sphere intersection, which will be used later to tell whether an acoustic ray passes through the microphone.

4.3.1 Ray-triangle intersection

For the ray-triangle intersection algorithm, an algorithm introduced by Tomas Möller and Ben Trumbore [32] is used. This algorithm is fast and uses minimum storage. It is generally implemented as:

```
rayTriangleIntersection(Triangle triangle, Ray ray) {
begin
        EPS = 0.000001f;
        double a, f, u, v;
        Vector3f s,h;
        Vector3f vertex0 = triangle.A;
        h.cross(ray.direction, triangle.edgeCA);
        a = triangle.edgeBA.dot(h);
        if (a < EPS)
                return -1;
        end
        if (Math.abs(a) < EPS)
                return -1;
        end
        f = 1 / a;
        s.set(ray.origin.x - vertex0.x,
        ray.origin.y - vertex0.y, ray.origin.z - vertex0.z);
        u = f * (s.dot(h));
        if (u < 0.0 || u > 1.0)
                return -1;
        end
        q.cross(s, triangle.edgeBA);
        v = f * ray.direction.dot(q);
        if (v < 0.0 || u + v > 1.0)
                return -1;
        end
```

```
        double t = f * triangle.edgeCA.dot(q);
        if (t > EPS)
                return (float) t;
        else
                return -1;
        end
end
```

4.3.2 Ray-box intersection

The ray-box intersection algorithm used in this work is an algorithm proposed by Smits [34] and implemented by Amy Williams, Steve Barrus, R. Keith Morley and Peter Shirley from University of Utah [33]. Inverse direction vector ought to be precalculated for each ray, as division is a computationally difficult operation.

```
rayAABBIntersection(Ray ray, AABB box)
begin
        double tmin, tmax, tymin, tymax, tzmin, tzmax;
        if (ray.direction.x >= 0)
                tmin = (box.xmin - ray.origin.x) * ray.dirDivision.x;
                tmax = (box.xmax - ray.origin.x) * ray.dirDivision.x;
        else
                tmin = (box.xmax - ray.origin.x) * ray.dirDivision.x;
                tmax = (box.xmin - ray.origin.x) * ray.dirDivision.x;
        end
        if (ray.direction.y >= 0)
                tymin = (box.ymin - ray.origin.y) * ray.dirDivision.y;
                tymax = (box.ymax - ray.origin.y) * ray.dirDivision.y;
        else
                tymin = (box.ymax - ray.origin.y) * ray.dirDivision.y;
                tymax = (box.ymin - ray.origin.y) * ray.dirDivision.y;
        end
        if ((tmin > tymax) || (tymin > tmax))
                ray.sect.x = 0;
                ray.sect.y = 0;
                return;
        end
        if (tymin > tmin)
                tmin = tymin;
        end
        if (tymax < tmax)
                tmax = tymax;
        end
        if (ray.direction.z >= 0)
                tzmin = (box.zmin - ray.origin.z) * ray.dirDivision.z;
                tzmax = (box.zmax - ray.origin.z) * ray.dirDivision.z;
        else
                tzmin = (box.zmax - ray.origin.z) * ray.dirDivision.z;
                tzmax = (box.zmin - ray.origin.z) * ray.dirDivision.z;
        end
        if ((tmin > tzmax) || (tzmin > tmax))
                ray.sect.x = 0;
                ray.sect.y = 0;
                return;
```

```
        end
        if (tzmin > tmin)
                tmin = tzmin;
        end
        if (tzmax < tmax)
                tmax = tzmax;
        end
        ray.sect.x = (float) tmin;
        ray.sect.y = (float) tmax;
end
```

4.3.3 Ray-sphere intersection

The last intersection algorithm that needs to be implemented is the ray-sphere intersection. This algorithm can easily be deduced using simple linear algebra and Pythagoras' theorem:

```
raySphereIntersection(Ray ray, Vector3d sphereOrigin, double r)
begin
        Vector3d connection;
        connection.sub(sphereOrigin, ray.origin);
        double tc = connection.dot(ray.direction);
       //check if sphere is in the direction of the ray
        if (tc < 0)
                return -1;
        end
       //ray-sphere intersection
        double dis = Math.sqrt(Math.abs(Math.pow(tc, 2) -
        Math.pow(connection.length(), 2)));
        /* distance can't be more than radius
         if it is there is no intersection */
        if (dis > r)
                return -1;
        end
        // returns the closest intersection to the ray origin
        return (tc - Math.sqrt(Math.pow(r, 2) - Math.pow(dis, 2)));

end
```

Chapter 5
Sensor modelling with ray-tracing

This chapter provides information about how cameras and acoustic sensors are modelled using ray tracing.

5.1 Camera

Modelling cameras with ray tracing is a very simple concept. We must create one ray for each pixel, which returns the object intersected by it and its distance or colour. Combined with the basic ray tracing algorithm we get:

```
for i = 0:cameraLength
        for j = 0:cameraWidth
                Ray ray = new ray from camera through pixelframe(j,i)
                for each object in scene do
                        if ray intersects object
                        and intersection is nearest so far then
                                record intersection distance and object
                end
        camImg(j,i) = nearest object's color
        j++
        end
i++
end
```

The only problem is finding the direction and origin of the ray in world coordinates. In order to correctly determine it, several transformation matrices have to be defined first. These transformations are necessary because a model space needs to be traced with a ray created in the image space (pixel positions).

5.1.1 Model Matrix

The first matrix is the Model Matrix. This matrix transforms model coordinates into the world space. It can be easily derived since it combines rotation, translation and scale. One rotation matrix is necessary for each axis the model was rotated over.

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where θ is the angle of rotation over the x axis.

$$R_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where θ is the angle of rotation over the y axis.

$$R_z = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where θ is the angle of rotation over the z axis.

$$T = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where x, y and z are the model coordinates in the world space.

$$M = STR_xR_yR_z$$

5.1.2 View Matrix

The second important camera matrix is the view matrix. This matrix transforms coordinates from the world-space to the camera view space - it's the inverse of the combined camera rotation and translation matrix. It can be derived from rotation and translation of the camera as:

$$V = (TR_xR_yR_z)^{-1}$$

5.1.3 Projection Matrix

The third matrix is the projection matrix. This matrix transforms the view space into the projection space. This is a transformation from 3D space to 2D and uses the homogenous coordinates to work properly. The projection matrix depends on the projection type of the camera. There are several types of projections with two being used the most often: the perspective projection and the ortographic projection. The perspective projection is used in this work.

The perspective projection matrix is not trivial to derive. The whole derivation can be found at [36]. The resulting matrix is:

$$P = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n\ over0} \\ 0 & -1 & 0 & \end{pmatrix}$$

where l,r,t,b,f and n are the 6 camera parameters left, right, top, bottom, far and near.

5.1.4 Projection Space to Image Space

Finally, the projection space (which has coordinates from (-1,-1) in the bottom-left corner to (1,1) in the top-right corner) needs to be transformed into the Image space with (0,0) in the top-left corner and (camera pixel width,camera pixel height) in the bottom-right corner. This can be done easily as:

$$(px, py) = (\frac{(x+1)w}{2}, h - \frac{(y+1)h}{2})$$

where

w	is the image's width in pixels
h	is the image's height in pixels
(x, y)	are the coordinates of a point in the projection space
(px, py)	are the coordinates of a point in the image space

5.1.5 Image to world space transformation

By taking the inverse of these matrices, we can easily transform the direction vector from image space to world space. When we have the ray set properly, ray tracing can begin.

5.2 Acoustic sensor modelling

In this work, the acoustics are represented as rays travelling through the environment. In order to successfully model an acoustic sensor, several principles need to be considered:

5.2.1 Sound absorption while travelling

Firstly, the energy of the sound is partially absorbed by the medium, when sound travels through it. This attenuation can be expressed as [35]:

$$W(L) = W_{ini}e^{-mL}$$

where

L	distance travelled
m	attenuation coefficient of the material
W_{ini}	initial power of the sound

or

$$W(L) = W_{ini} - mL$$

in decibels.

The attenuation coefficient depends upon the frequency of the sound. For air, this coefficient varies from 0 dB/m at 10 Hz and lower and roughly 0.1 dB/m at 10000 Hz [35]. These values were determined at an air temperature of $20°C$ and 60% humidity.

5.2.2 Surface-related sound absorption

Secondly, sound energy is absorbed when sound reaches a surface. Only a portion of the sound is reflected back, some of it is absorbed by the object that was hit. The degree of absorption depends on the angle at which the sound hits the surface (angle of incidence) [35].

$$W(L) = W_{ini}(1 - \alpha)$$

$$\alpha = \alpha(\theta)$$

where

α	absorption coefficient of the surface
θ	angle of incidence
W_{ini}	initial power of the sound

5.2.3 Sound reflection and refraction. Energy distribution functions.

Lastly, when sound hits a surface, it is necessary to calculate the angle of reflection and refraction and determine the distribution function from them.

For the reflected part of the sound, the law of reflection says that the angle of incidence is equal to the angle of reflection.

For the refracted part of the sound, we will need to determine the angle of refraction using Snell's law. This law states can be written as formula:

$$\frac{\sin\theta_{refraction}}{\sin\theta_{incidence}} = \frac{v_{refracted}}{v_{incidence}}$$

Consequently, we only need to determine the velocities and the angle of incidence.

With these angles we can determine an estimation of the real distribution function. The distribution function consists of the specular part (when the angle is close to the precise reflection angle) and the diffraction part (the scattering of sound in all directions).

In the end, the distribution function is only dependent on the angle of incidence, absorption coefficient, the velocities in the media and the ratio of diffraction.

Chapter 6
Implementation

This chapter provides information about the implementation of the program with emphasis on the optimalisation that had to be done after implementing the program for the first time. It also talks about the architecture chosen for this program.

6.1 Architecture

The system has been implemented according to the following design:

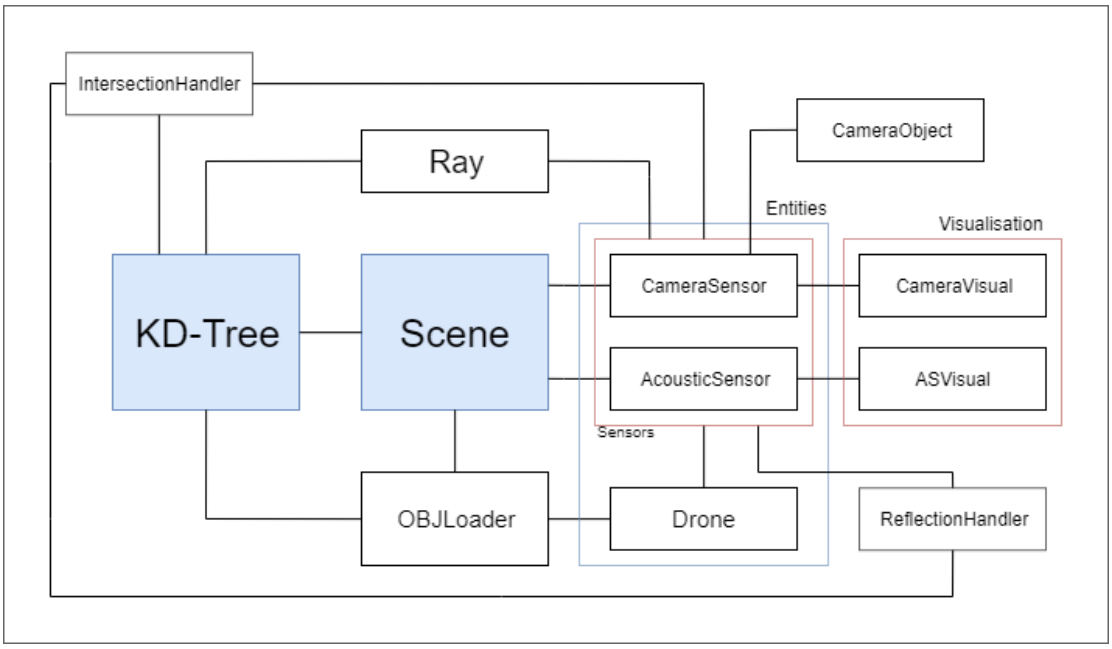


Figure 6.1. Architectural design

The design does not include geometrical primitives such as vectors, matrices, triangles, boxes and spheres, used by multiple classes, as this would make the design unclear. The implementation works with vectors and matrices to represent all the positions, directions and transformations. For this, it is recommended to use the vecmath library. All the algorithms have been implemented as proposed in the last two chapters, however, some improvements had to be introduced in the implementation to significantly speed up the computations and to make real-time detection possible.

6.2 Implemented classes and improvements

With the heavy use of objects, especially vectors, it is necessary to avoid overusing the Java garvage collector. To save memory, objects that are of no further use to the program are collected and destroyed. To avoid overloading the collector, fields have

been widely used in the implementation. With their addition, a lot of the objects need to be created only once and instead of recreating them a new value is set to them when necessary. The downside to them is that they make the classes unsafe to use in parallel computing. This can be avoided by synchronizing the classes.

6.2.1 KD-Tree

The designed building and tracing algorithms in the previous chapter are very effective. They do not, however, talk about the data representation of the tree. In this implementation, nodes are represented as 64-bit integers. This makes the memory load very fast, which results in fast tracing. To make this possible, the data in the node needs to be compressed and written using bit shifting. To cut down on stored data, each node's left child is stored right after it in the array. Therefore:

$$index_{left} = index_{parent} + 1$$

Thanks to this rule, only the index of the right child has to be stored in the node. For tracing, we must also store the coordinate that was used to split and the value of the splitting plane on said coordinate.

2 bits are used to store the coordinate, 31 bits are stored for the right child node index and 30 bits for the dimension value of the splitting plane. For scenes that consist of fewer triangles, and with lower desired precision, 32-bit integers could be used for performance enhancement.

6.2.2 CameraSensor, CameraObject and CameraVisual

The CameraObject class is only used to generate matrices necessary for scene viewing. The class is a part of the AgentFly visualisation framework.

The CameraSensor is the main class that handles detection. It computes views and passes the data to the CameraVisual object. CameraVisual uses the JFrame to visualise the data.

The main improvement to the Camera was the addition of significant pixels. These pixels are calculated from the bounding box of the drone, and only these pixels need to be recomputed on each frame. The cameras therefore work as follows:

Firstly, the static view is computed by tracing the world (its KD-tree) with a ray for each pixel. The direction of this ray is provided by the CameraObject through vector transformation.

```
for (int i = 0; i < dim.height; i++) {
        for (int j = 0; j < dim.width; j++) {
                pixelPos.set(j, i);
                camera.getRay(pixelPos,
                 camera.getPosition(), direction);
                Ray r = new Ray(camera.getPosition(), direction);
                float inter = tree.treeTraversal(r).x;
                staticView[j][i] = inter;
                if (max < inter) {
                        max = inter;
                }
                if (min > inter) {
                        min = inter;
                }
        }
}
```

This view needs to be computed only once, unless the objects in the environment are moved or the camera is moved.

Next, the maximum and minimum significant pixels are derived by transforming the corners of the drone's bounding box to the projection space of the camera. To achieve this, we must transform each corner with the MVP matrix. This matrix combines the projection, view and model matrices. The core of this algorithm is therefore:

```
for (Vector4d cor : box.corners) {
        corner.set(cor);
        MVP.transform(corner);
        corner.scale(1 / corner.w);
        minC.x = Math.min(minC.x, corner.x);
        maxC.x = Math.max(maxC.x, corner.x);
        minC.y = Math.min(minC.y, corner.y);
        maxC.y = Math.max(maxC.y, corner.y);
        minC.z = Math.min(minC.z, corner.z);
        maxC.z = Math.max(maxC.z, corner.z);
}
```

In case the drone is not in the frame, an outOfFrame field is set to true and the function is returned. After dealing with this case, the maximum and minimum pixel indices are found as:

```
wmax = (float) (maxC.x + 1) * dim.width / 2;
hmax = (float) (dim.height - (minC.y + 1) * dim.height / 2);
wmin = (float) (minC.x + 1) * dim.width / 2;
hmin = (float) (dim.height - (maxC.y + 1) * dim.height / 2);
```

These pixels are derived on each frame of the camera's video. Now for each of these significant pixels a ray traces the drone's inner tree and the distance to the intersection with the drone is returned. For this to work, the position and direction of the ray need to be transformed into the drone's model space. This distance is compared to the distance stored in the static view, and the shorter is chosen. If the drone is closer, a hit is recorded. These values are stored in the dynamic view and passed on to the CameraVisual, which handles visualisation through the JFrame class as an icon. The pixel values are directly converted from the distance of the object, therefore more distant objects are a lighter blue. With enough hits recorded, it is decided that the drone would be detected by a real camera. The amount of pixels necessary for detection has been calibrated to 12. This value needs to be calibrated for each camera that is to be simulated, depending on its specifications.

Since only a small part of the frame is redrawn for each frame, the camera model works very fast, especially if the drone is far away from it.

6.2.3 AcousticSensor

The main speed-up of the AcousticSensor is achieved through two very similar methods listen and listenFully. The listen method was designed to quickly detect if the drone is detectable by the sensor by exiting early upon detection. The listenFully method keeps tracing rays until all rays drop below a threshold. This allows the visualisation of the detection and the computing of an estimation of the energy cumulated on the

sensor. With these two methods, it can be chosen whether the detection should be fast or thorough.

This class uses a LinkedList object to store rays, that are to be traced. Each ray is traced until it hits a triangle in the KD-tree. If the ray also intersects the sphere of the sensor, the distances to these intersections are compared. If the sphere intersection distance is shorter, the hit is recorded and the energy at the point of intersection is added to the estimated cumulated energy. If the triangle is closer, or the sphere intersection returns no intersection, the ray is reflected and refracted using the ReflectionGenerator class from the util package. This class generates new rays and adds them to the LinkedList. If at any point the energy of the traced ray drops below a set threshold, this ray is discarded and another ray is taken from the queue.

If desired, each ray's parameters are recorded for visualisation. The visualisation framework used is a part of the AgentFly visualisation framework.

6.2.4 OBJLoader

To obtain the triangle mesh, an OBJLoader needed to be implemented. The implemented OBJLoader is a simple translator of .obj and .mtl files into arrays of vertices and indices that can be used to create Triangle objects from them. The OBJLoader is a part of the AgentFly visualisation framework.

6.2.5 Ray and AcousticRay

The ray class has been implemented according to its previous design. Apart from the origin and the direction of the ray an inverse of its direction is also precomputed and stored. This needs to be done as division is computationally demanding and the inverse is needed multiple times. The returned intersection, when the ray hits an object, is stored in a field of type Vector2f which can be used to store either two intersection distances or an intersection distance and the ID of the intersected object. The class AcousticRay that extends the ray class stores two energies - one as a threshold for the simulation and one for counting the cumulative energy on the sensor. It also stores the material that the ray goes through as per the design.

6.2.6 AABB

The AABB class represents an axis-aligned bounding box. It contains 6 fields of type float containing the minimum and maximum values on each of the three axes. It was improved by adding one field of type Vector4d that is only initialized by some constructors. These vectors represent the corners of the box and can be easily transformed with projection and view matrices for camera visualisation.

6.2.7 Triangle

The Triangle class represents a geometrical triangle. It stores its 3 points as Vector3f objects but it also stores two edges from one point to two others, so they do not have to be computed each time the triangle is intersected. Each triangle also has an ID, unique to this triangle in one data structure. A triangle can also store its material necessary for calculating reflections.

6.2.8 IntersectionHandler

The proposed intersection algorithms from previous chapters have been implemented in this class.

6.2.9 Drone

Another class representing an entity is the Drone class. An instance of this class stores the drone's velocity, the model transformation matrix and the KD-Tree that contains its object. All the movement of the drone is done through transforming its model matrix. For this rotation and movement methods are implemented, for example:

```
public void rotDroneX(double angleVelo, double t) {
        temp.setIdentity();
        temp.rotX(angleVelo * t);
        M.mul(M, temp);
}
```

The drone also has a method for creating acoustic rays on a sphere with uniform distribution, representing the sound it makes.

6.2.10 Settings

The Settings class and its 3 extensions: CameraSettings, AcousticSettings and DroneSettings represent settings of entities in the environment. Each one has a method for returning a new instance of the object it sets. It is implemented for the purpose of easy swapping between settings. The extensions will be described in the next chapter.

6.2.11 ReflectionGenerator

The ReflectionGenerator stores methods for refraction and reflection used when a triangle is hit by a ray. The acousticDistribution method is the main method for generating rays. They are generated on a uniformly distributed sphere. It is checked multiple times whether a ray's energy dropped under the threshold to avoid unnecessary computations. This check is done when the ray hits a triangle and when each new ray is generated. Before generating random rays, one ray in the direction of reflection and one ray in the direction of the the sensor is created to prioritize rays with higher chance of reaching the sensor.

Chapter 7
Experiments

This chapter describes several experiments and their results. The correct implementation will be verified, as well as the possibility of detecting real drones with acoustic sensors and cameras. Custom scenario generation is discussed at the very end of this chapter.

7.1 Experiment 1 - Closed space scenario

The first experiment is performed in a closed space of a church. The purpose of this experiment is to visualise all the acoustic reflections that can be traced in such an environment. The experiment is set with following parameters:

World model	Sibenik.obj (130 000 triangles)
Background noise	20 dB
Drone model	LinkQuad.obj (60 cm x 40 cm x 30 cm)
Drone sound level	55 dB
Drone velocity	5 m/s
Cameras	1
Camera Resolution	800 x 800 pxs
Camera FOV	0.6918 rad
Acoustic Sensors	1
Initial No. of acoustic rays	10
Acoustic reflections per hit	10
Acoustic visualisation	True

In this experiment, the drone is in a static position 10 meters away from the sensors.

7.1.1 Results

The drone is detected by both the acoustic sensor and the camera. The visualisation can be seen in Figure 7.1. and Figure 7.2.

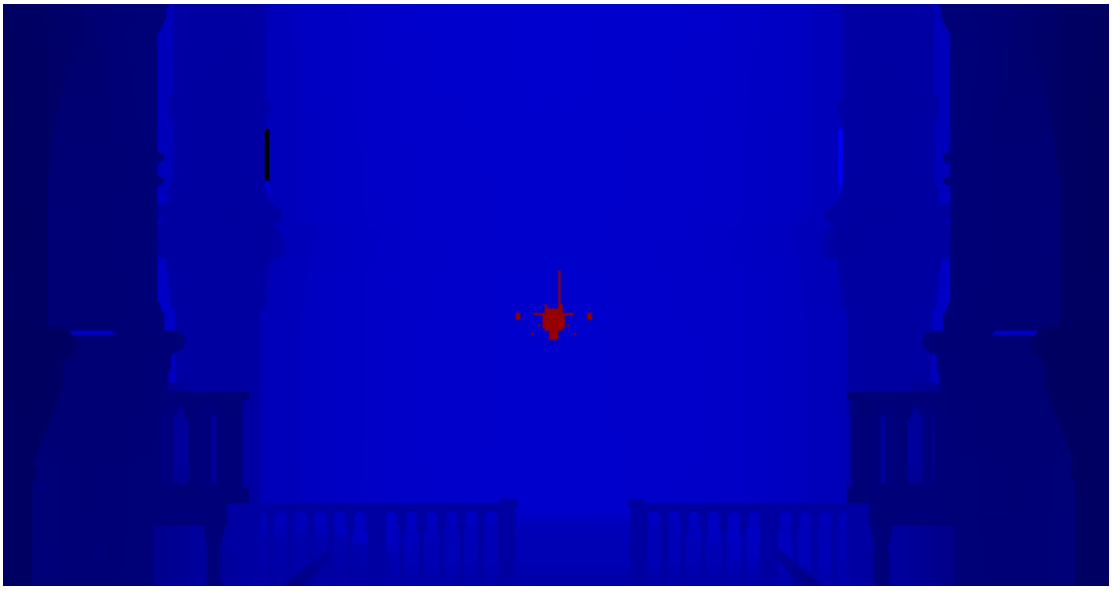


Figure 7.1. A video shot capturing the drone

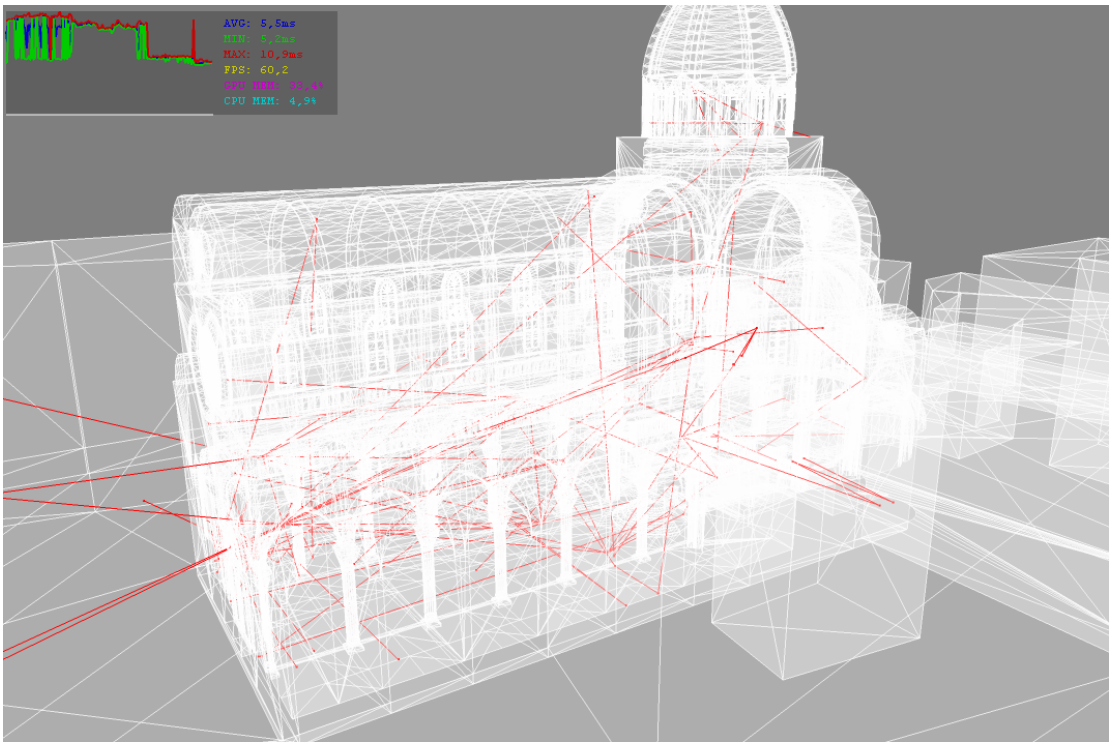


Figure 7.2. Acoustic sensor testing: 10x10 rays

Both sensors seem to be working properly, so the implementation is verified. It can be seen that 10 initial rays with 10 reflections do not fill the environment completely, but the visualisation is clear. Both parameters are now set to 20.

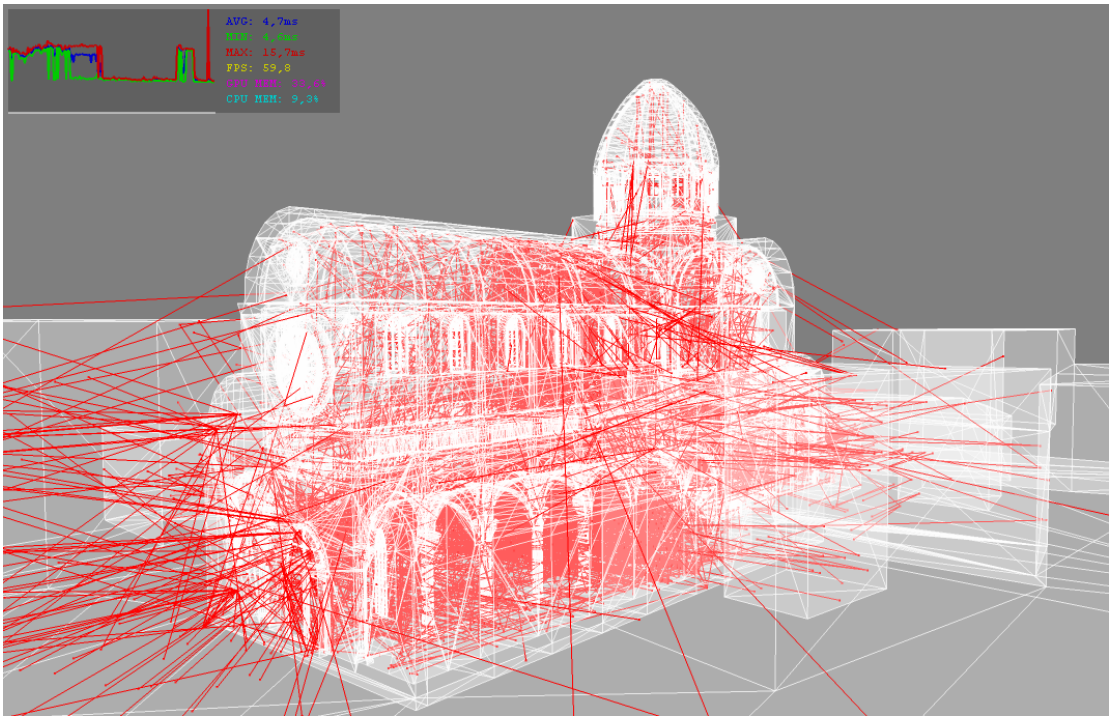


Figure 7.3. Acoustic sensor testing: 20x20 rays

With this many rays, the simulation is more precise, but the visualisation is very confusing. For further testing, a high number of rays generated will be chosen, but the visualisation will be omitted.

7.2 Experiment 2 - Camera max distance scenario

The second scenario sets a camera on a hill. The camera is directly aimed at the drone. The purpose of this simulation is to determine the maximum distance at which a camera can detect a drone. The simulation will be run multiple times to determine the relation between the camera's FOV and the distance at which the drone is detected. It can be assumed that the lower FOV camera will detect the drone further away, as more pixels will be detecting the drone.

World model	object_right.obj (65 000 triangles)
Drone model	LinkQuad.obj (60 cm x 40 cm x 30 cm)
Drone velocity	5 m/s
Cameras	1
Camera Resolution	1920 x 1080 pxs
Camera FOV	$(\frac{\pi}{6}, \frac{\pi}{5}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{\pi}{2})$ rad

7.2.1 Result

The results of the simulation can be seen in the following figure:

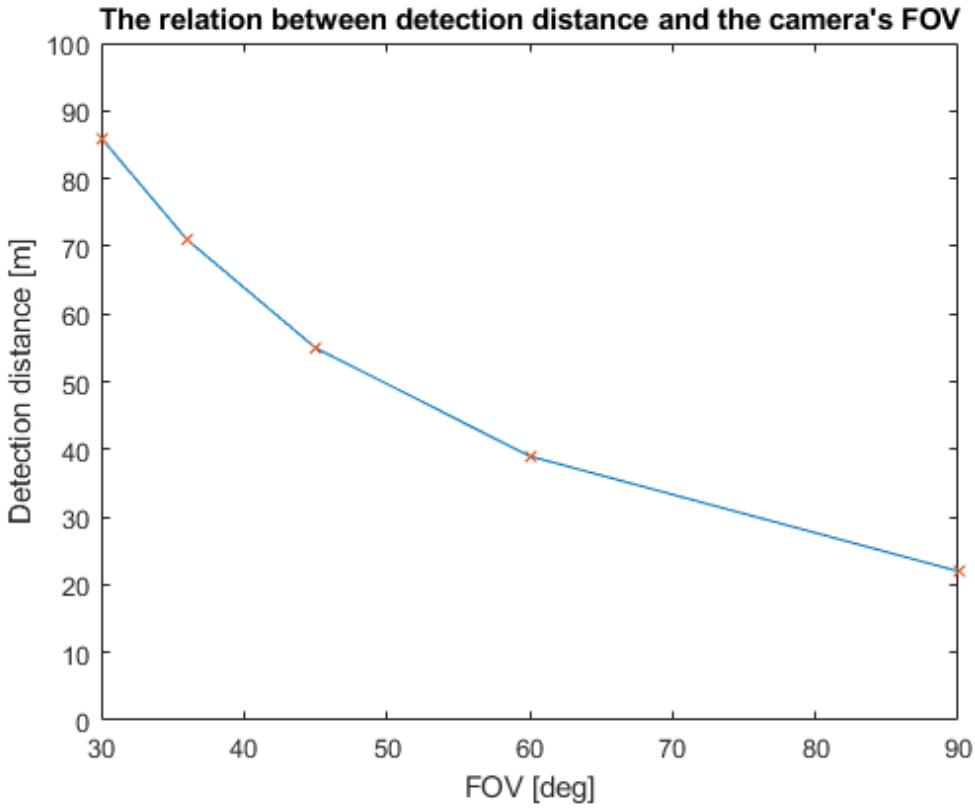


Figure 7.4. The relation between drone's distance and the FOV of the detecting camera

The results of this experiment confirm the assumption that lower FOV cameras detect objects further. Therefore, to boost detection range of a camera system, multiple lower FOV cameras should be used. The detection distance is heavily dependent on the threshold function. This function for now is a static number of pixels, which is far from ideal. However, determining a stable threshold function for detection depending solely on the pixels would require testing with real drones, cameras and detection algorithms and a substantial amount of time. For now, calibrating the function for each simulated camera so that the simulation yields similar results to the camera's specifications should suffice.

7.3 Experiment 3 - Acoustic sensor max distance scenario

The third scenario puts the sensor onto a hill. The purpose again is to determine the maximum distance of the sensor. The way the acoustic sensor detection is set up in this project causes that if the drone has no obstacles between it and the sensor detection depends only on the difference between the power of the sound emitted by the drone and the background noise in the scene. It is assumed that the detection distance can be calculated as:

$$L = (P_{drone} - P_{BG} - 15dB)m_{air}$$

where

L	maximum detection distance
P_{drone}	the power of the sound emitted by the drone
P_{drone}	background noise level
m_{air}	attenuation coefficient of air at the drone's sound frequency

It has been stated that the attenuation coefficient depends on the frequency of the sound; therefore to flawlessly simulate the detection the sound's frequency spectrum would have to be measured. For now, the coefficient has been set to 0.05 dB/m. Again the simulation will be run multiple times, decreasing the background noise each time by 5 dB.

World model	object_right.obj (65 000 triangles)
Background noise	30 - 5n dB
Drone model	LinkQuad.obj (60 cm x 40 cm x 30 cm)
Drone sound level	50 dB
Drone velocity	5 m/s
Acoustic Sensors	1
Initial No. of acoustic rays	20
Acoustic reflections per hit	20
Acoustic visualisation	False

7.3.1 Results

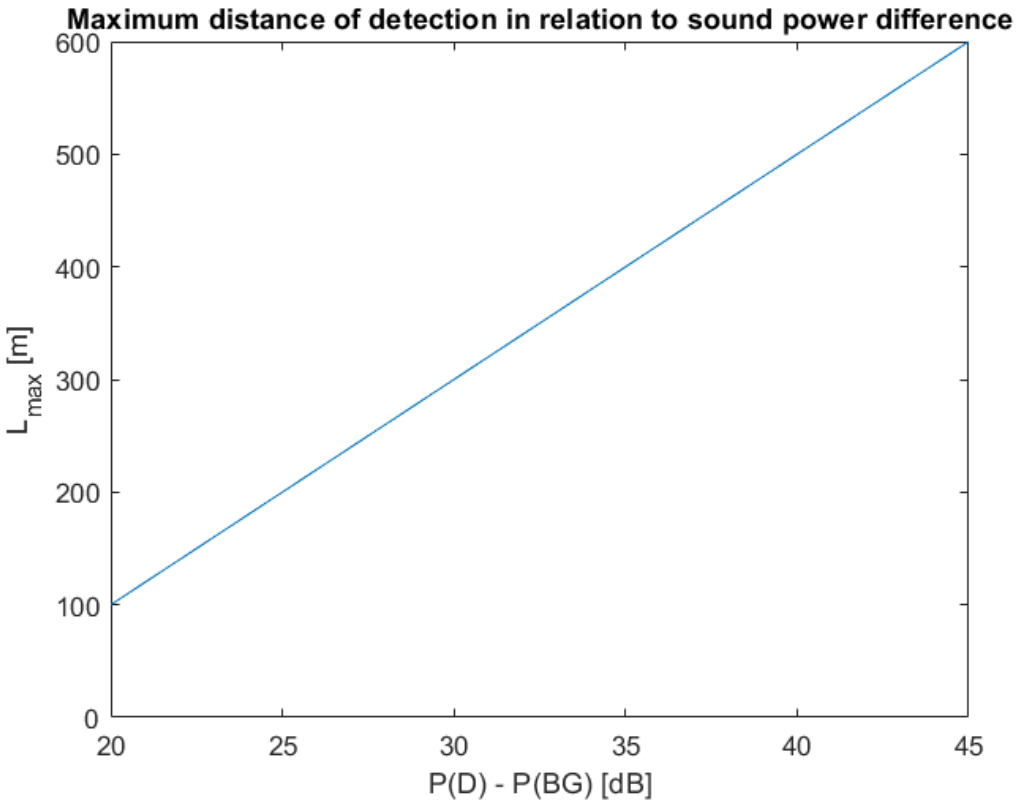


Figure 7.5. The relation between the difference of the bg noise and the drone's noise level, and the distance of detection

The results confirm the assumed dependency.

7.4 Experiment 4 - Multiple-sensor scenario

The last predesigned scenario utilises multiple sensors to cover an object completely. The object is a house with a rectangular platform with three cameras in each corner of its roof. An outline of this area can be seen in Figure 7.6.

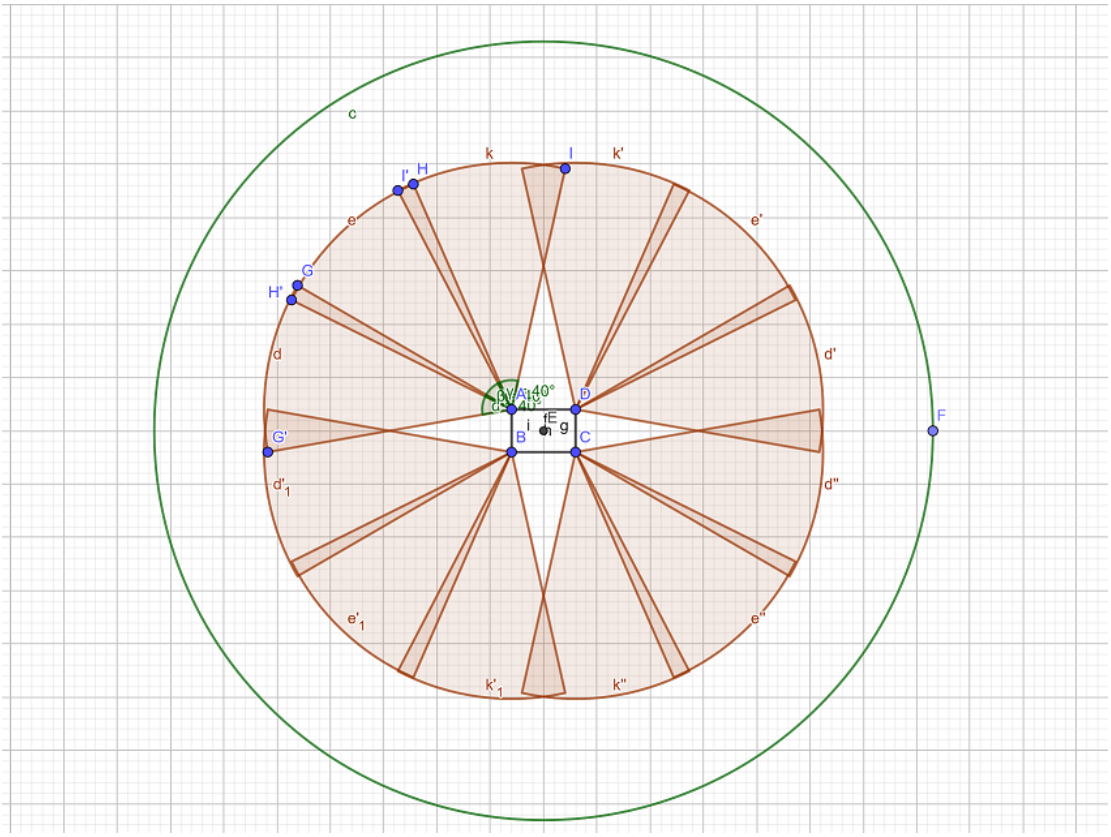


Figure 7.6. The covered area

An acoustic sensor is placed on the roof, in its middle. The acoustic sensor's purpose is to cover the area if the drone flies too high, undetected by the cameras. This scenario serves as an example of area coverage with multiple sensors, as well as for testing of simulating 13 sensors at once. This scenario is meant to be adjusted; it is possible to quickly change the drone's movements, the power of the sound it emits, the drone's model to a smaller or bigger size, etc. The background noise has been set to 32 dB to represent a quiet rural area. The results are derived from following parameters:

World model	object_right.obj (65 000 triangles)
Background noise	32 dB
Drone model	LinkQuad.obj (60 cm x 40 cm x 30 cm)
Drone sound level	52 dB
Drone velocity	5 m/s
Cameras	12
Camera Resolution	1920 x 1080 pxs
Camera FOV	0.6918 rad
Acoustic Sensors	1
Initial No. of acoustic rays	20
Acoustic reflections per hit	20
Acoustic visualisation	True

7.4.1 Results

The results of the simulations indicate a maximum detection range of 100 meters. This has bad implications for multiple attack types, especially if a small drone's purpose was to film an area from a distance, as it would go undetected, unless more effective sensors are set up, or they are set up further from the object. This would increase the number of sensors necessary to cover the area and the overall cost of the system. Bigger and noisier drones might be detected in a sufficient range and therefore giving enough time to react to the attack. However, as research showed, these drones are often used to carry explosives or weapons, so a swifter response to the attack is necessary. In conclusion, it is almost always necessary to set up a larger array of sensors around the object to detect drone attacks at sufficient distances.

7.5 Designing a sensor area coverage model for real infrastructure

This section discusses the design of a multi-sensor coverage of an area and the suggested approach derived from the simulations.

- Determining the properties of the layout - this includes the background noise, considering all the possible camera obstructions or sources of noise.
- Setting requirements for detection
- Choosing sensors satisfying the requirements - this includes the price of the sensors, as well as its detection power. Sometimes the size of the sensor also needs to be considered, not to disrupt the surrounding environment.
- Designing a coverage model using geometry
- Calibrating sensor detection functions in the environment according to real values
- Simulating attacks on the infrastructure in the simulation environment

7.6 Generating custom scenarios

To generate a custom scenario, it is recommended to create a new extension of the abstract class scene. Simply use the load function of the Scene class to load all the necessary objects, and the Settings class to set up all the sensors in the environment. It is recommended to store of the same type in an array, and iterate over it. This makes the code very clear. When the sensors are set, the drone can be moved around the simulated area, as described earlier.

Chapter 8
Conclusion and Future work

8.1 Future work

With no access to real drones and sensors for detection and with limited time, the simulation environment proposed in this work comes with options of improvement to it.

Firstly, radars should be added to the sensor models, as they also are a powerful means of drone detection, as described in the research.

As far as acoustics are concerned, many parameters are heavily dependent on the frequency of the sound. Therefore it would be beneficial to implement this in the simulations. However, there are hardly any articles or databases that the drone's sound spectrum could be derived from, therefore it would be best to measure these values and make a new database for it.

Another database that would improve the simulation would be a table of materials, from which all the parameters could easily be fetched. So far a generic `solid` material is used for the objects in this work, and a constant value of absorption independent of the angle. This could potentially cause deviation from real sound propagation, however mostly only when the drone is behind a thick layer of material.

With the camera, a better detection threshold function can be implemented. This would require a deep study of detection algorithms used for drone detection and finding correlation between the number and arrangement of pixels and the detections using statistics.

Finally the biggest drawback of the camera is the weather and light conditions. It would be very difficult to implement this, since simulating weather conditions in an image is an area still explored in computer vision. With the colour visualisation set up and with access to a real-time detection algorithm, the simulation could generate real-looking pictures, that would be analyzed by the detector.

In its current state the simulation environment has too many flaws to solely depend on it when testing an area coverage model. It does, however, provide an option to coverage models very fast and discard designs that have no chance of working.

8.2 Conclusion

Systems with both acoustic and camera sensors are able to detect most types of attacks. However, these systems can be avoided, by using larger drones with both propellers and gliders. These drones can ascend using the propellers, turn them off and glide into the destination. This way, it is able to go above the camera's field of view and make little sound. They, however, require very specific design and cannot be stopped without being

detected. Consequently, their main use would be to carry explosives as they are unable to capture stable video.

In conclusion, unless a drone is specifically design to fly over cameras without making sound, it will be detected by multi-sensor systems. The detection range is longer if the drone is larger or noisier.

References

[1] NATO Science & Technology Organization, 9th NATO Military Sensing Symposium, STO-MP-SET-241, May 2017.

[2] Department of Homeland Security, Unmanned Aircraft Systems Addressing Critical Infrastructure Security Challenges, Fact Sheet, February 2017.

[3] Gabriel C. Birch, John C. Griffin, and Matthew K. Erdman.: UAS Detection, Classification, and Neutralization: Market Survey 2015, Sandia National Laboratories, 2015.

[4] Custers, Bart, Drone Technology: Types, Payloads, Applications, Frequency Spectrum Issues and Future Developments (2016). Vergouw B., Nagel H., Bondt G. & Custers B.H.M. (2016), Drone Technology: Types, Payloads, Applications, Frequency Spectrum Issues and Future Developments. In: Custers B.H.M. (red.) The Future of Drone Use. Heidelberg: Springer.. Available at SSRN: https://ssrn.com/abstract=3047760

[5] An-18-year-old-mounted-a-gun-to-a-drone-and-fired-shots-in-the-middle-of-the-woods - Business Insider. [online] Available at: http://www.businessinsider.com/an-18-year-old-mounted-a-gun-to-a-drone-and-fired-shots-in-the-middle-of-the-woods-2015-7

[6] US firm reveals gun-toting drone that can fire in mid-air - BBC News. [online] Available at: http://www.bbc.com/news/technology-40901393

[7] A defense company put a machine gun on a drone — Tech Crunch . [online] Available at: https://techcrunch.com/2017/08/17/a-defense-company-put-a-machine-gun-on-a-drone/

[8] Ten sentenced for smuggling drugs into prisons by drones - BBC News. [online] Available at: http://www.bbc.com/news/uk-42341416

[9] Mexican drug cartels using drones to smuggle heroin, meth, cocaine into U.S. - Washington Times. [online] Available at: https://www.washingtontimes.com/news/2017/aug/20/mexican-drug-cartels-using-drones-to-smuggle-heroi/

[10] List of countries where drones are banned or restricted — DJI FORUM. [online] Available at: https://forum.dji.com/thread-98600-1-1.html.

[11] M. I. Skolnik. Radar Handbook, Third Edition. New York: The McGraw-Hill Companies, 2008.

[12] P. Poitevin, M. Pelletier and P. Lamontagne, “Challenges in detecting UAS with radar,” 2017 International Carnahan Conference on Security Technology (ICCST), Madrid, Spain, 2017.

[13] J. Drozdowicz et al., “35 GHz FMCW drone detection system,” 2016 17th International Radar Symposium (IRS), Krakow, 2016.

[14] AUDS (Anti UAV Defence System) Fact Sheet. http://liteye.com/assets/auds-fact-sheet-10-20162.pdf

[15] SpotterRF radars systems for perimeter surveillance. [online] Available at: https://anti-drone.eu/products/perimeter-surveillance-radars.

[16] Long-range sensors for UAS acoustic detection. [online] Available at: https://anti-drone.eu/products/acoustic-sensors/long-range-acoustic-sensors.html.

[17] DroneSentry — DroneShield. [online] Available at: https://www.droneshield.com/sentry.

[18] Wang, Bang. Coverage Control in Sensor Networks. London: Springer-Verlag London, 2010. Print.

[19] J. Vilímek and L. Buřita, "Ways for copter drone acustic detection," 2017 International Conference on Military Technologies (ICMT), Brno, 2017, pp. 349-353.

[20] I. Güvenc, O. Ozdemir, Y. Yapici, H. Mehrpouyan and D. Matolak, Detection, localization, and tracking of unauthorized UAS and Jammers, 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), St. Petersburg, FL, 2017, pp. 1-10.

[21] M. Benyamin and G. H. Goldman, "Acoustic Detection and Tracking of a Class I UAS with a Small Tetrahedral Microphone Array," Army Research Laboratory Technical Report (ARL-TR-7086), DTIC Document, Tech. Rep., Sep. 2014.

[22] J. Lemley, S. Bazrafkan, and P. Corcoran, "Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision." IEEE Consumer Electronics Magazine, vol. 6, no. 2, pp. 48–56, April 2017.

[23] Red.com. (2017). Introducing the RED EPIC-W And New WEAPON Cameras. [online] Available at: http://www.red.com/news/introducing-epic-w-and-weapon-8k [Accessed 31 Dec. 2017].

[24] H. Liu, Z. Wei, Y. Chen, J. Pan, L. Lin and Y. Ren, "Drone Detection Based on an Audio-Assisted Camera Array," 2017 IEEE Third International Conference on Multimedia Big Data (BigMM), Laguna Hills, CA, 2017, pp. 402-406.

[25] APPEL A.: "Some techniques for shading machine renderings of solids." In Proceedings of the April 30–May 2, 1968, spring joint computer conference (New York, NY, USA, 1968), AFIPS '68 (Spring), ACM, pp. 37–45.

[26] Andrew S. Glassner. An Introduction to Ray Tracing. Academic Press, 1989.

[27] Henrik Wann Jensen, Pet Christensen. "High quality rendering using ray tracing and photon mapping," SIGGRAPH '07 ACM SIGGRAPH 2007 courses. San Diego, California, 2007.

[28] I. Wald and V. Havran, "On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)," 2006 IEEE Symposium on Interactive Ray Tracing, Salt Lake City, UT, 2006, pp. 61-69.

[29] BENTLEY J.: "Multidimensional binary search trees used for associative searching." Commun. ACM 18 (1975), 509–517

[30] J. David MacDonald and Kellogg S. Booth. "Heuristics for ray tracing using space subdivision. Visual Computer," 6(6):153–65, 1990.

[31] Hapala, Michal and Vlastimil Havran. "Review: Kd-tree Traversal Algorithms for Ray Tracing." Comput. Graph. Forum 30 (2011): 199-213.

[32] Akenine-Möller, Tomas and Ben Trumbore. "Fast, minimum storage ray/triangle intersection." SIGGRAPH Courses (1997).

[33] A. Williams, S. Barrus, R. K. Morley, P. Shirley. “An Efficient and Robust Ray-Box Intersection Algorithm.” J. Graphics Tools. 2005.

[34] Peter Shirley. “Fundamentals of Computer Graphics.” AK Peters, 2002.

[35] Elorza, David Oliva. “Room acoustics modeling using the ray- tracing method: implementation and evaluation.” (2006).

[36] OpenGL Projection Matrix. [online] Available at: http://www.songho.ca/opengl/gl_projectionmatrix.html.