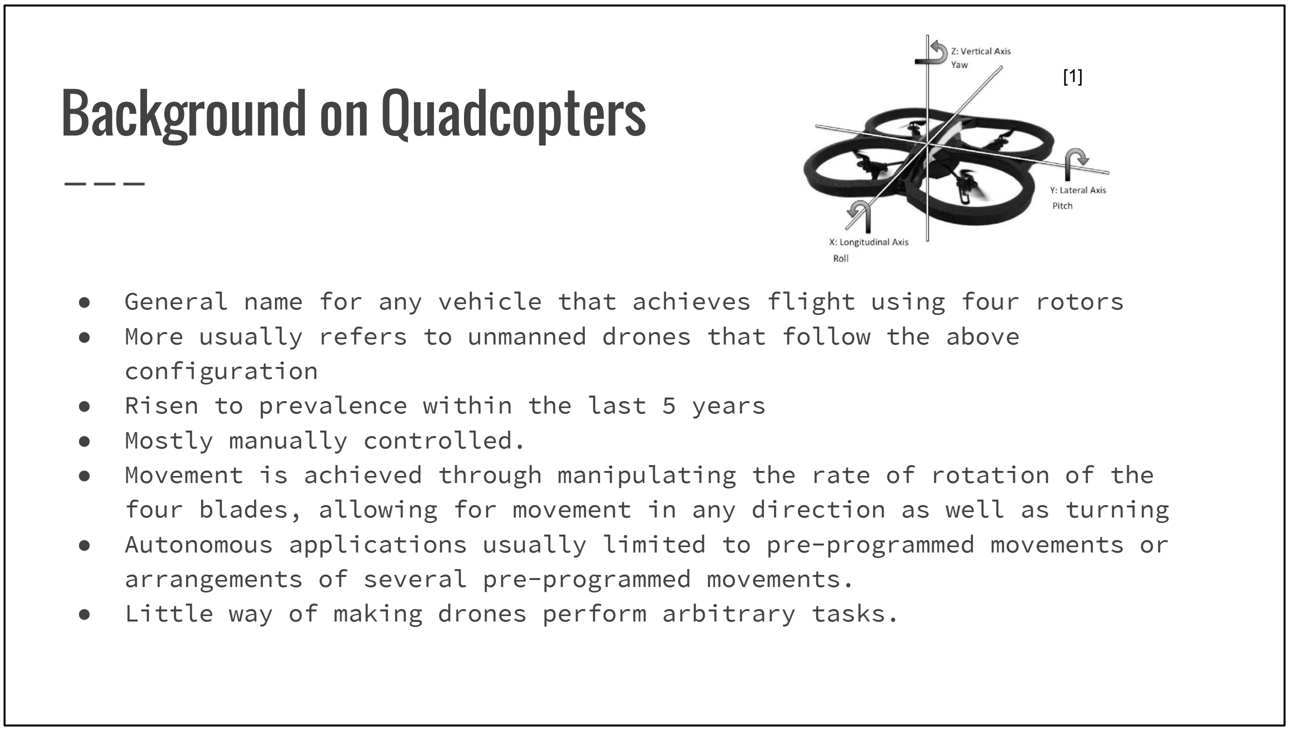
OCTODRONE

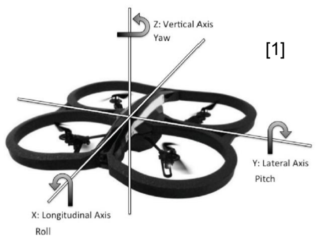
Simulation and Deployment for Autonomous Drone Networks

Overview

- Background on quadcopters
- Project introduction
- The simulation software
- octoDrone demonstration
- Research implications
- Deployment demonstration
- Outreach implications
- Wider uses & further work
- Questions



Background on Quadcopters

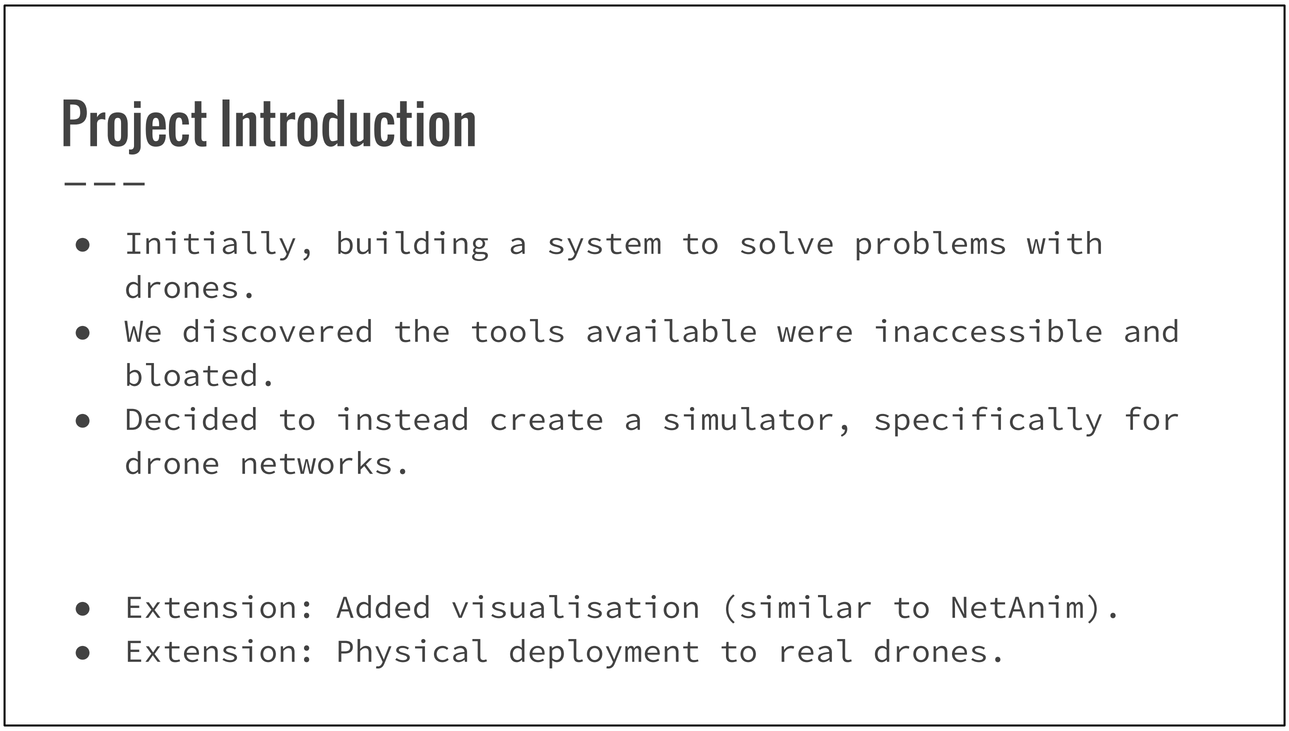


- General name for any vehicle that achieves flight using four rotors
- More usually refers to unmanned drones that follow the above configuration
- Risen to prevalence within the last 5 years
- Mostly manually controlled.
- Movement is achieved through manipulating the rate of rotation of the four blades, allowing for movement in any direction as well as turning
- Autonomous applications usually limited to pre-programmed movements or arrangements of several pre-programmed movements.
- Little way of making drones perform arbitrary tasks.

Some more information on how quadcopters move:

Quadcopters move by inclining themselves relative to the horizon. This is achieved by having two of the four coplanar rotors spinning clockwise, and two spinning anticlockwise. The differing direction of rotation is to ensure that the torque (turning force) exerted by the spinning of the rotors sums to zero. This is achieved in helicopters by the use of a tail rotor. Quadcopters are underactuated in as much as they have fewer motors than they have degrees of freedom (four rotors, six degrees of freedom), meaning that they cannot follow arbitrary paths in 3D space. Put another way, there are some movements that drones are unable to achieve, such as simultaneously moving and turning, which directly stem from the lack of additional rotors.

Moving forward entails having one of the rotors which is spinning in a certain direction (say clockwise) move faster and the the opposing rotor of the same direction move slower. This has the effect of tilting the drone on the Y axis. With the drone thus inclined, the air displaced by the rotors now causes it to move forward. The same technique can be used in reverse to move the drone backwards, and applied to the rotors spinning in the opposite direction (in this case anticlockwise) to bank the drone left or right on the X axis. Finally, a drone is turned on the Z axis by increasing the rotation speed (and thus torque) of a pair of rotors operating in the same direction. With the torque in one direction greater than the other the drone begins to turn. Altitude is controlled by increasing or decreasing the rotational speed of all motors in unison.



Project Introduction

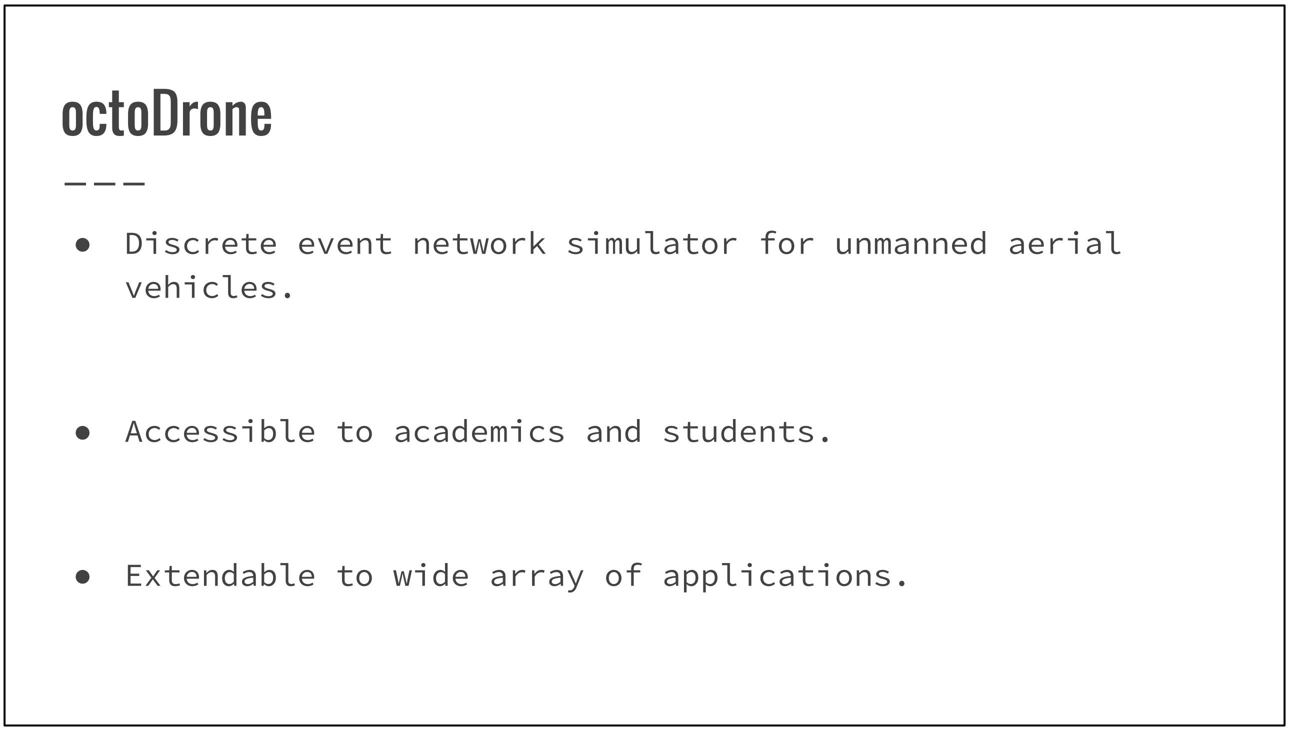
- Initially, building a system to solve problems with drones.
- We discovered the tools available were inaccessible and bloated.
- Decided to instead create a simulator, specifically for drone networks.

- Extension: Added visualisation (similar to NetAnim).
- Extension: Physical deployment to real drones.

Initially, the goal of the project was to create a system which would use drones to solve interesting problems, exploring what was possible with a drone sensor network.

The intention was to use an existing product such as ns3, but we discovered that all of the tools available were inaccessible, bloated, or did not have very good support for quadcopters.

To solve this, we changed the scope of the project so that we could focus on creating a domain specific simulator for drone networks (or any flying sensor platforms). We later extended our original solution with a visualisation system similar to NetAnim (which is used by ns3), and to make the programs that the simulator ran available to use on real drones. We then created a reference deployment on the department's Parrot quadcopters which we will show later on in the presentation.



octoDrone

- Discrete event network simulator for unmanned aerial vehicles.
- Accessible to academics and students.
- Extendable to wide array of applications.

octoDrone:

- Discrete Event Network Simulator
- Specifically designed for unmanned aerial vehicles
- Allows for the implementation of any functionality the user wishes to give it.
- Given the time it takes to learn ns-3, we wanted to make something easier to learn in a short amount of time for academics and students

Domain specific for academics and students, unlike ns-3 and other tools.

The Team

Will Seymour - Project Manager, communications, and physical deployment.

Alex Henson - Coordinating the report and research.

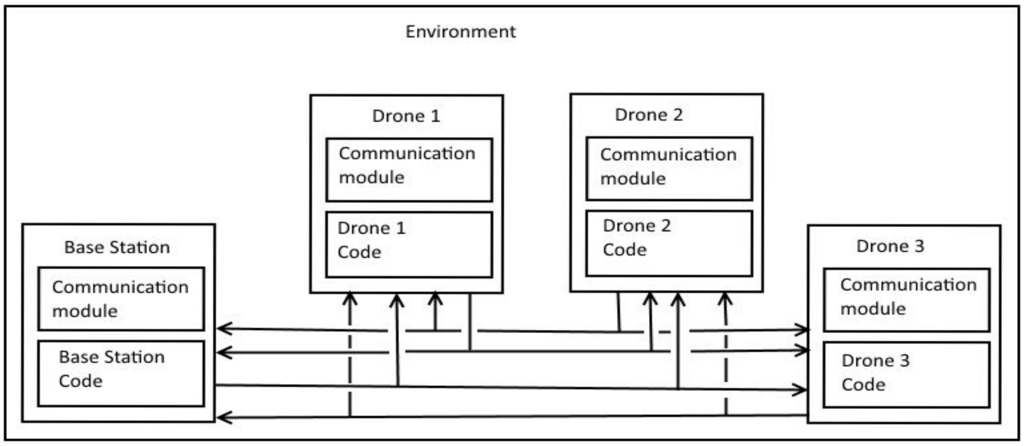
Ben De Ivey - Developing the demonstration simulation.

Jon Gibson - Developing the simulator.

Simulation Software: Overview

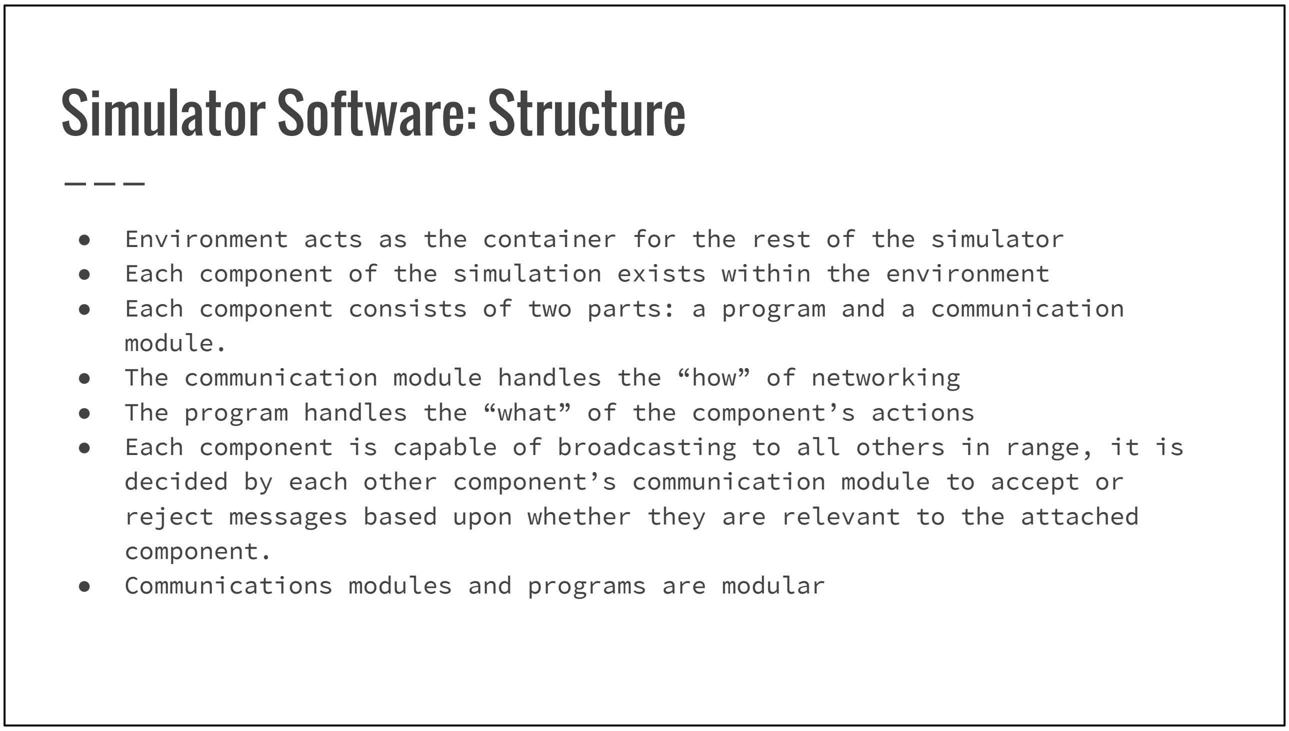
- Written in C++
- Focus on emulating multiple systems through the use of threads.
- Minimal feature set so arbitrary drone programs can be run on the simulator
- Primary focus was flexibility and extensibility.

Simulation Software: Structure Example



Key points from the diagram:

- Everything is contained within the simulation environment
- Nodes can communicate with all other nodes in the simulation
- Functionality is separated into a program (or code) and a communications module

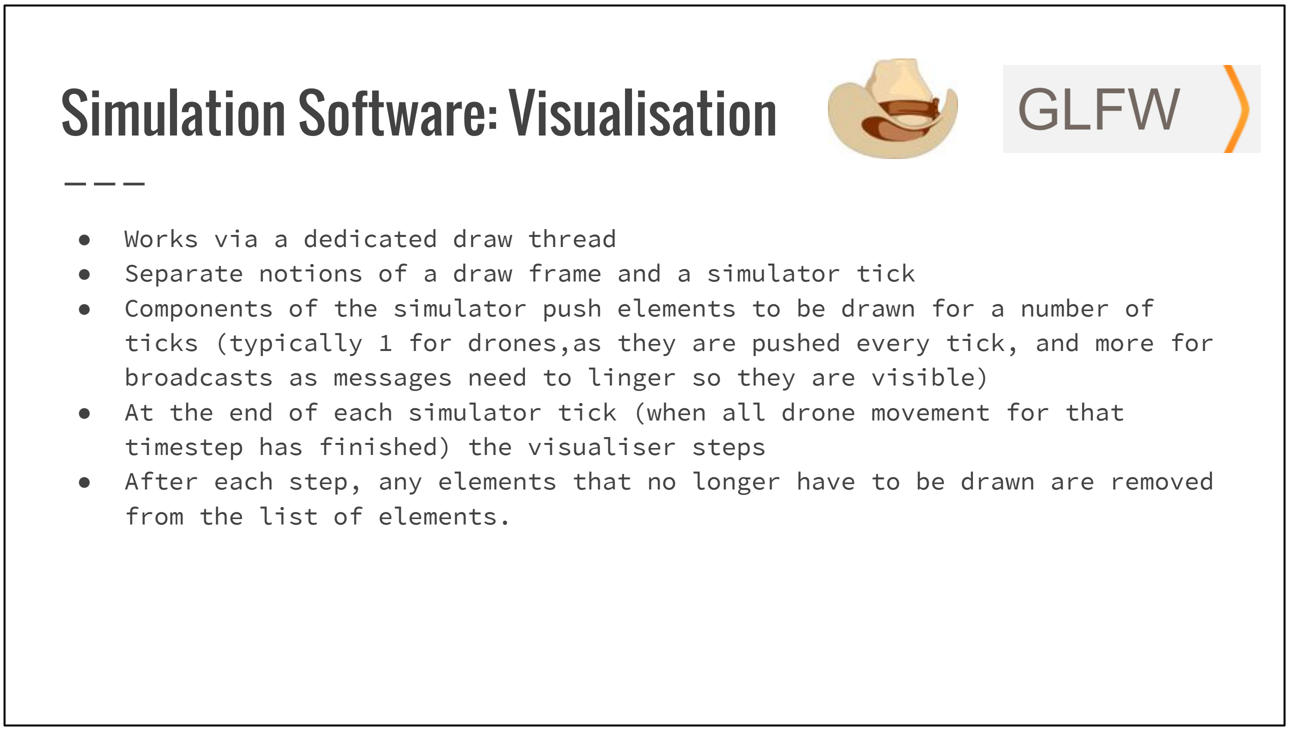


Simulator Software: Structure

- Environment acts as the container for the rest of the simulator
- Each component of the simulation exists within the environment
- Each component consists of two parts: a program and a communication module.
- The communication module handles the “how” of networking
- The program handles the “what” of the component’s actions
- Each component is capable of broadcasting to all others in range, it is decided by each other component’s communication module to accept or reject messages based upon whether they are relevant to the attached component.
- Communications modules and programs are modular

The simulator was created to be as modular as possible with each subsystem having very clearly defined links to the other parts of the simulator (for instance the methods of “commMod” that interact with environment). Using this design allows for far easier modification and testing of each part of the system, as each part presents a contract with the other pieces; so long as that contract remains unchanged, the other parts of the system should be able to continue working identically even if the other components are changed. This also has benefits to possible optimisations of the system as any optimisations will be trivially integrable so long as they do not violate this contract.

The simulator is also extensible. This is actually the key to the way that the simulator works: the point of the simulator is that users extend the classes provided to use the simulator. This also allows users to extend the simulator in order to alter the way that the simulator works. As is, the simulator is devoid of any non-essential features, but this is by design; the main problem identified with ns-3 was feature bloat, causing the entire system to be unfocused and thus more difficult to develop for than a more focussed system.



Simulation Software: Visualisation

- Works via a dedicated draw thread
- Separate notions of a draw frame and a simulator tick
- Components of the simulator push elements to be drawn for a number of ticks (typically 1 for drones,as they are pushed every tick, and more for broadcasts as messages need to linger so they are visible)
- At the end of each simulator tick (when all drone movement for that timestep has finished) the visualiser steps
- After each step, any elements that no longer have to be drawn are removed from the list of elements.

The visualisation system provided a large set of implementation challenges. Because of the multi-threaded nature of the simulator, different threads can push elements to be visualised at any point. These element pushes are mainly handled through several helper functions that each call an underlying element pushing function with preset arguments. This simplifies both the implementation and use of these functions. The issue with this method is that the element list can be polled to change at any time, even when the visualiser is in the middle of a drawing operation. Due to the way C++ iterators work, and the fact the list has elements removed periodically (due to the fact that these elements are removed when they have exceeded their lifespan), this could cause iterators previously pointing to valid elements to become invalid.

The solution to this problem was to lock both the step function and the draw function behind the same mutex lock. Effectively, this makes the two functions mutually exclusive, allowing only one of the two functions to be running at any given time. This causes any changes to the list of elements to be done only when there are not any active pointers to the list.

octoDrone Demonstration

Creating a Program: API

Send a message:	`void send_message(Message* contents);`
Blocking Receive:	`Message* wait_for_message();`
Non-blocking receive:	`void receive_message(std::string contents);`

Get position:	`double getX();`
	`double getY();`
	`double getZ();`
	`double getSpeed();`
	`double getAngle();`

Movement:	`void turn(double angle);`
	`void move(Direction direction, double speed, double distance);`

Feedback:	`bool isAlive();`
	`bool has_finished_moving();`
	`double sense(std::string type);`

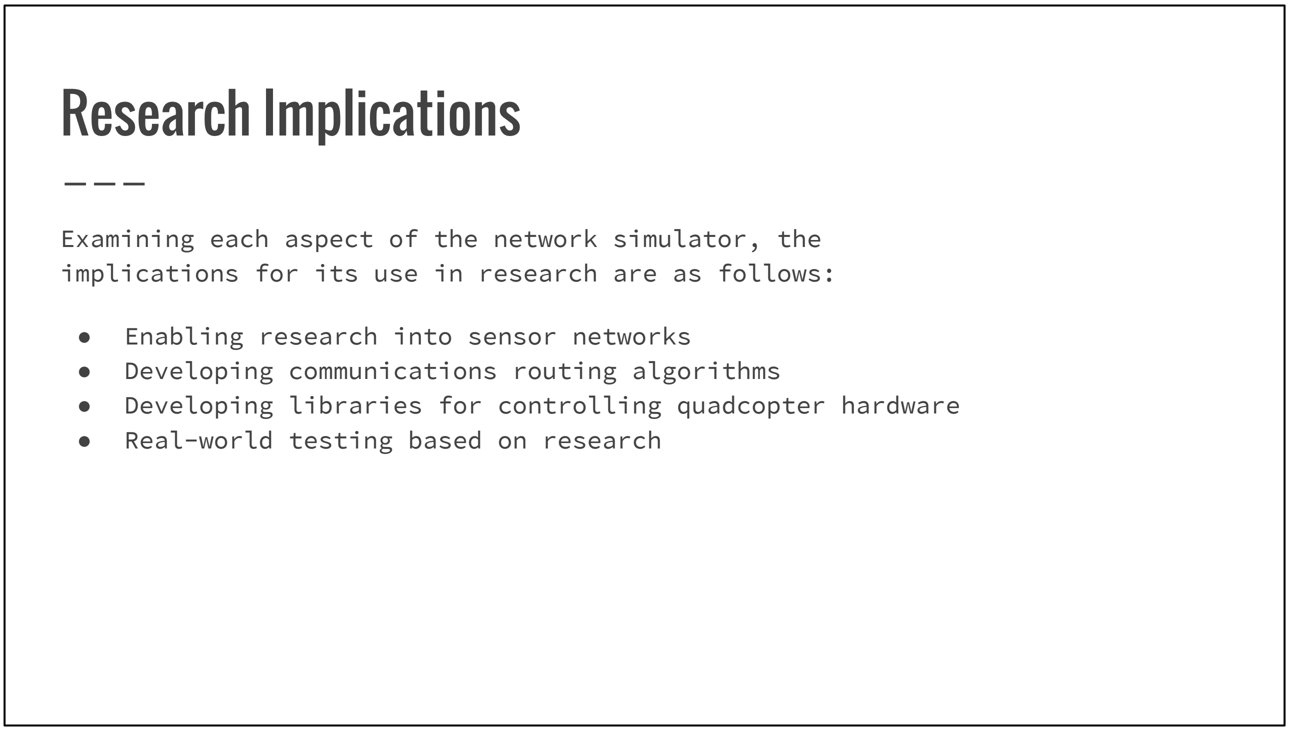
User implemented:	`virtual bool message_callback(Message* message) = 0;`
	`virtual void run() = 0;`

Creating a Simulation

```
std::map<std::string, data_type>* sensor_map = new std::map<std::string, data_type>;
Environment* env = new Environment(*sensor_map, 1.0, false);
std::atomic_flag stdout_lock = ATOMIC_FLAG_INIT;
IpAllocator allocator = IpAllocator(10, 0, 0, 1);


CommMod* comm_basic1 = new Basic_addressed(env, &stdout_lock, allocator.next());
CommMod* comm_basic2 = new Basic_addressed(env, &stdout_lock, allocator.next());
CommMod* comm_basic3 = new Basic_addressed(env, &stdout_lock, allocator.next());
CommMod* comm_basic4 = new Basic_addressed(env, &stdout_lock, allocator.next());
CommMod* comm_basic_base = new Basic_addressed(env, &stdout_lock, "10.0.0.255");

SensingBaseStation* basestation = new SensingBaseStation(comm_basic_base, 0.0, 0.0, 0.0, 1.0, 1.0, 10.0, 10.0);
env->setBaseStation(basestation);

SensingDrone* drone1 = new SensingDrone(comm_basic1, 1.0, 1.0, 0.0, 0.1, 0.5, env, false);
SensingDrone* drone2 = new SensingDrone(comm_basic2, 2.0, 2.0, 0.0, 1.0, 0.5, env, true);
SensingDrone* drone3 = new SensingDrone(comm_basic3, 2.0, 3.0, 0.0, 1.0, 0.5, env, true);
SensingDrone* drone4 = new SensingDrone(comm_basic4, 2.0, 4.0, 0.0, 1.0, 0.5, env, true);

env->addDrone(drone1);
env->addDrone(drone2);
env->addDrone(drone3);
env->addDrone(drone4);

env->run();
```

Command Line Output

```
basic_addressed@10.0.0.1:  waiting for new area
basic_addressed@10.0.0.4:  waiting for new area
basic_addressed@10.0.0.3:  waiting for new area
basic_addressed@10.0.0.2:  waiting for new area
Num Drones: 4
basic_addressed@10.0.0.1: NEWAREA=1.000000,1.000000,3.250000,10.000000
basic_addressed@10.0.0.1: Exiting
basic_addressed@10.0.0.4: NEWAREA=3.250000,1.000000,5.500000,10.000000
basic_addressed@10.0.0.4: Exiting
basic_addressed@10.0.0.255:  Received point (1.000000, 1.000000, 1.000000, 42.000000 from drone 10.0.0.1
basic_addressed@10.0.0.255:  Received point (1.000000, 2.000000, 1.000000, 42.000000 from drone 10.0.0.1
basic_addressed@10.0.0.255:  Received point (1.000000, 3.000000, 1.000000, 42.000000 from drone 10.0.0.1
basic_addressed@10.0.0.255:  Received point (1.000000, 4.000000, 1.000000, 42.000000 from drone 10.0.0.1
basic_addressed@10.0.0.255:  Received point (1.000000, 5.000000, 1.000000, 42.000000 from drone 10.0.0.1
basic_addressed@10.0.0.255:  Received point (1.000000, 6.000000, 1.000000, 42.000000 from drone 10.0.0.1
basic_addressed@10.0.0.255:  Received point (1.000000, 7.000000, 1.000000, 42.000000 from drone 10.0.0.1
```



Research Implications

Examining each aspect of the network simulator, the implications for its use in research are as follows:

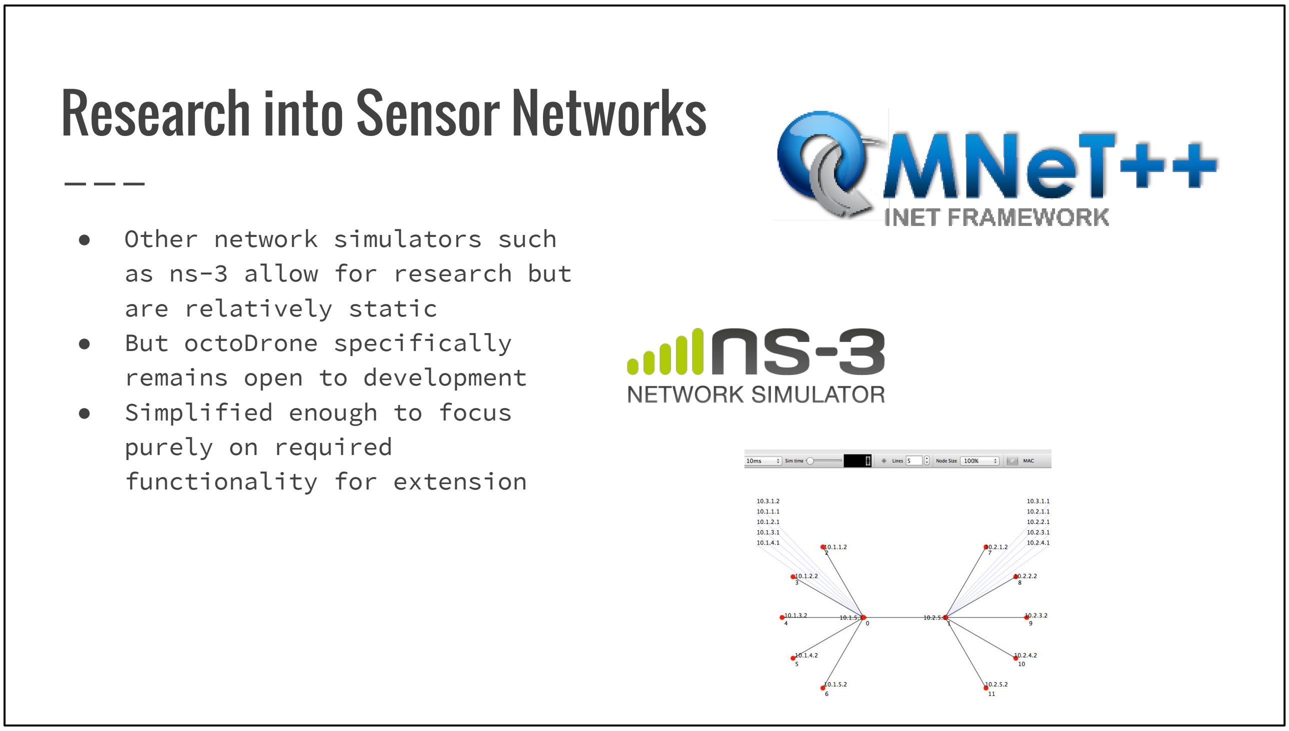
- Enabling research into sensor networks
- Developing communications routing algorithms
- Developing libraries for controlling quadcopter hardware
- Real-world testing based on research

In this section, I’d like to examine the research implications for octoDrone. We have mentioned that one of the target audiences for octoDrone is academics researching sensor networks, but how will it actually impact them?

The most obvious example is that our simulator allows for research into sensor networks. This includes how they operate, as well as the algorithms which govern their communication. There are currently a number of other products which achieve this goal.

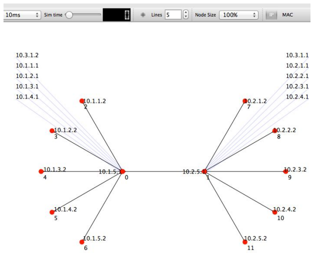
Where octoDrone begins to differ, however, is where it offers an opportunity for research staff to test their work on actual hardware in order to gain authentic results about real-world performance. This is something that is patchy across existing products, and helps to set octoDrone apart from the competition.
In addition to this, we have created a platform which both facilitates new ways to target quadcopter devices as well as allowing for easy integration of solutions which already exist (like ar-drone). This could be a useful way of uniting disparate implementations produced by the quadcopter community into something that is more accessible due to it being easier to get working.

In fact, this makes



Research into Sensor Networks

- Other network simulators such as ns-3 allow for research but are relatively static
- But octoDrone specifically remains open to development
- Simplified enough to focus purely on required functionality for extension



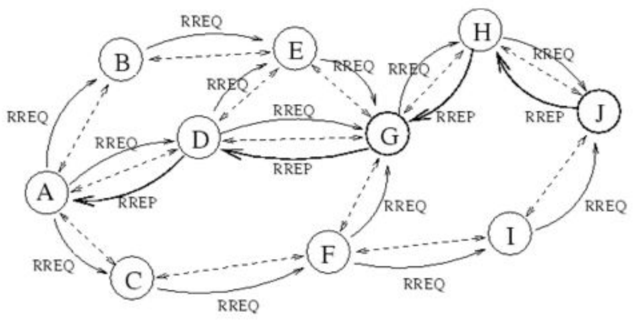
Network simulators such as ns have existed for a long time, and are well known throughout the community. When researching into existing sensor networks, it was found that, previously, 95% of all network simulation-based projects used ns-2 as the base library for development. With the introduction of its successor, ns-3 became the de facto choice for network simulation libraries. Other frameworks such as OMNet++ and GloMoSim also exist, and most, if not all, of these simulators are free and open source.

However, it remains a constant problem that, despite being open source, such frameworks are relatively static, such that they are not in active development. In addition, the level of functionality provides incredibly versatility, but the barrier to entry for learning how to use, much less contribute to development, remains a significant problem in developing simulations tailored specifically to the target domain.

On the other hand, octoDrone specifically remains open to development, and allows for a much easier transition into research into a particular problem space. The simulator is generic enough to be easily adaptable, whilst also providing only the level of functionality required for the task at hand, with the ability to be extended as required.

Developing Communications Routing Algorithms

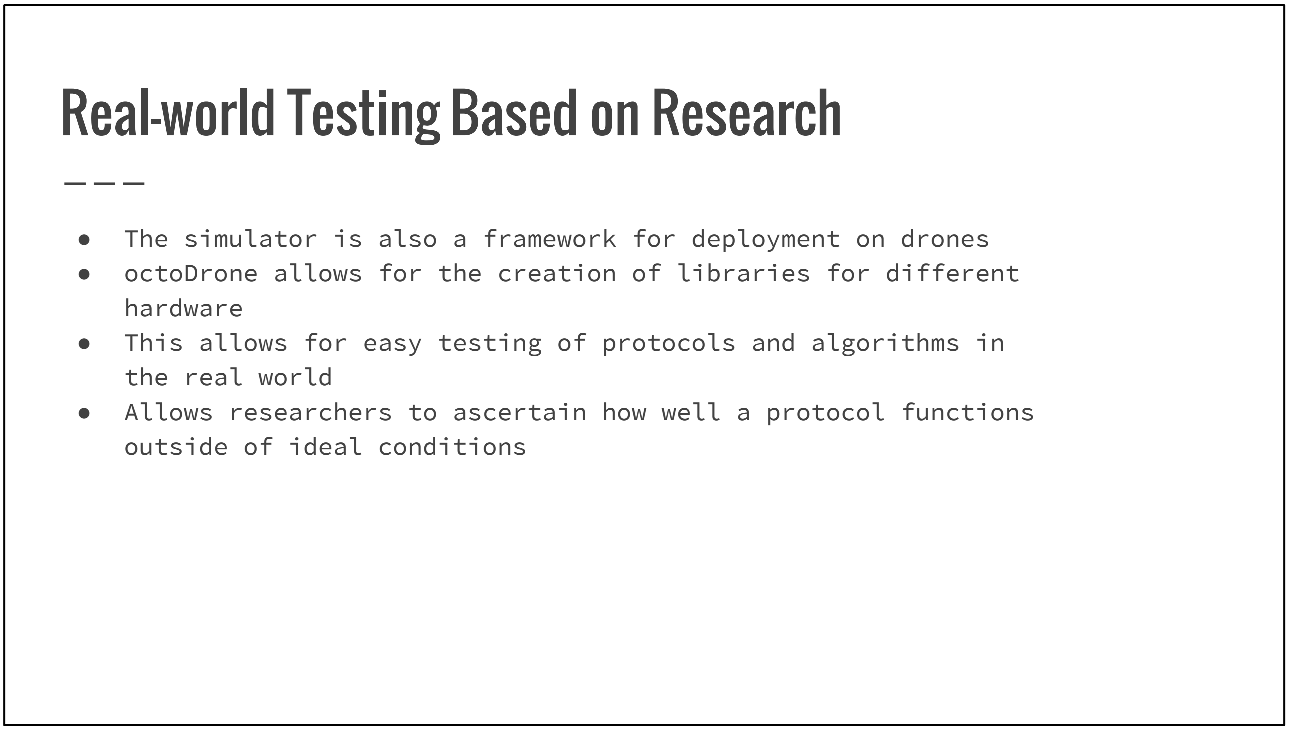
- Simulator comes with default library currently, but can be expanded easily (modularity)
- Libraries are low-level enough to precisely specify requirements
- Lightweight enough to make even complex simulations fast



The current simulator comes with a default library, which contains the basic messaging system, a more advanced basic messaging system with addressing, and an implementation of AODV, Ad Hoc on Demand Distance Vector Routing. One of the main strengths of octoDrone is the ability to arbitrarily add new algorithms to this library, and then choose the desired routing algorithm from among those included in the library. Although this will be addressed later, considerations for future work include the addition of more algorithms for increased variety.

The libraries are also low-level enough to precisely specify what functionality is needed. While other network simulators provide high-level function calls to specific algorithms, octoDrone provides a framework which is low-level enough for control but high level enough from the perspective of a program, and doesn’t require an extreme level of technical knowledge to apply the required functionality.

OctoDrone is also incredibly lightweight, and is designed with a level of simplicity which makes complex simulations much faster. In contrast, other simulators such as ns-3 require an adverse length of time to compile and run due to the amount of functionality included in the library.



Real-world Testing Based on Research

- The simulator is also a framework for deployment on drones
- octoDrone allows for the creation of libraries for different hardware
- This allows for easy testing of protocols and algorithms in the real world
- Allows researchers to ascertain how well a protocol functions outside of ideal conditions

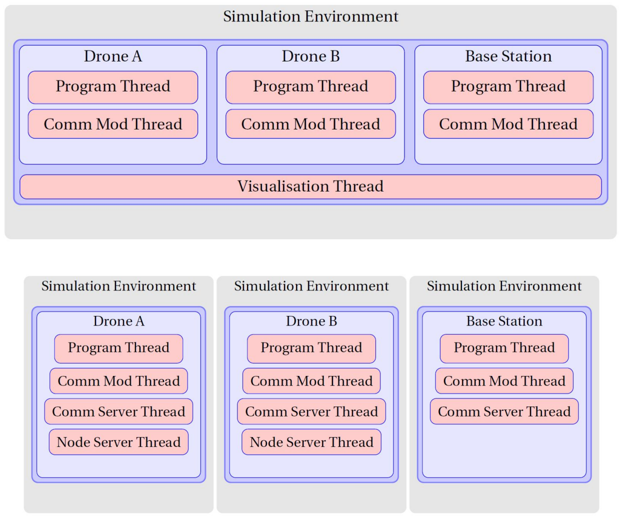
The simulator is also a framework for deployment on drones. What this means is that octoDrone will hook into calls to the simulator and replace them with calls to the hardware. From the perspective of the program, nothing changes, because of the extra processes that approximate simulator functions.

This means that, provided with the necessary libraries for different hardware, which researchers would be able to develop, octoDrone could be easily ported to other UAVs or networks with completely different specifications. Protocols and algorithms can be simulated, and then reflected in real-world situations

This makes it easy to test research in the real world, without having to rely solely on modelling the functionality of a protocol through simulation alone, where environmental conditions may affect results.

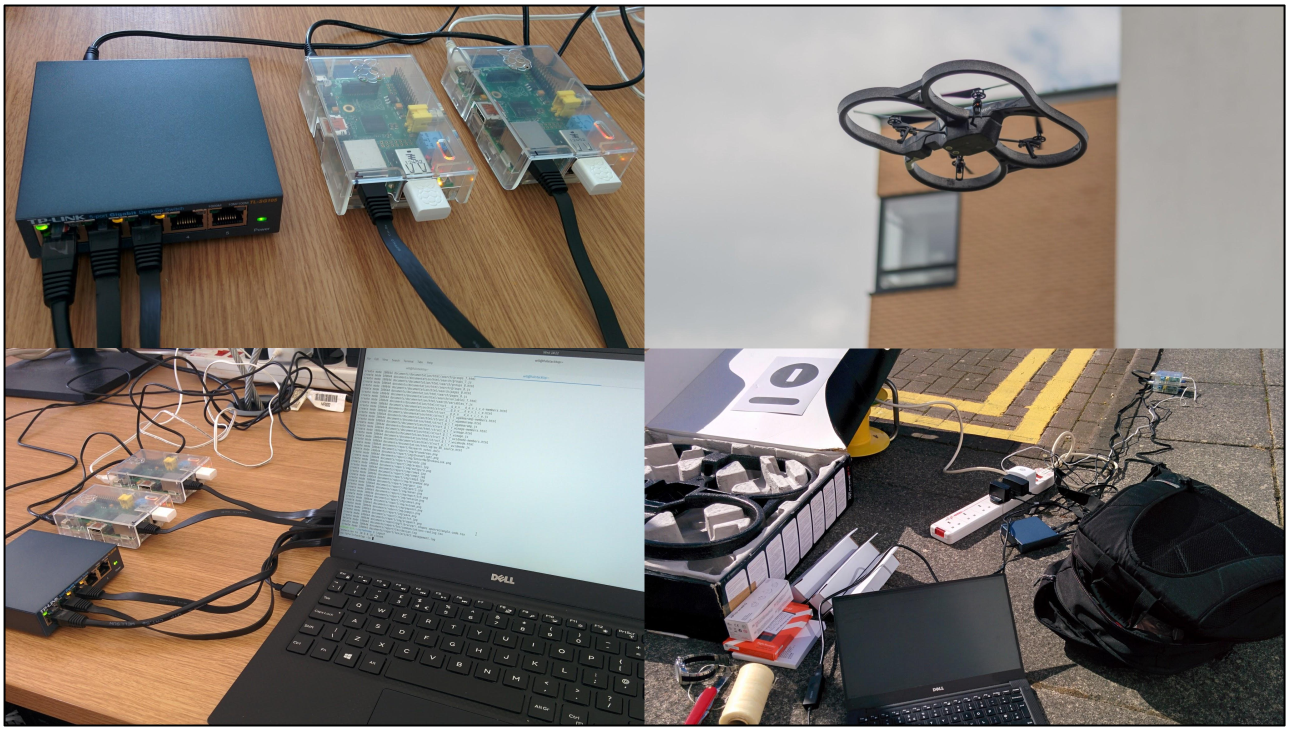
Deployment Demonstration

Architecture Comparison



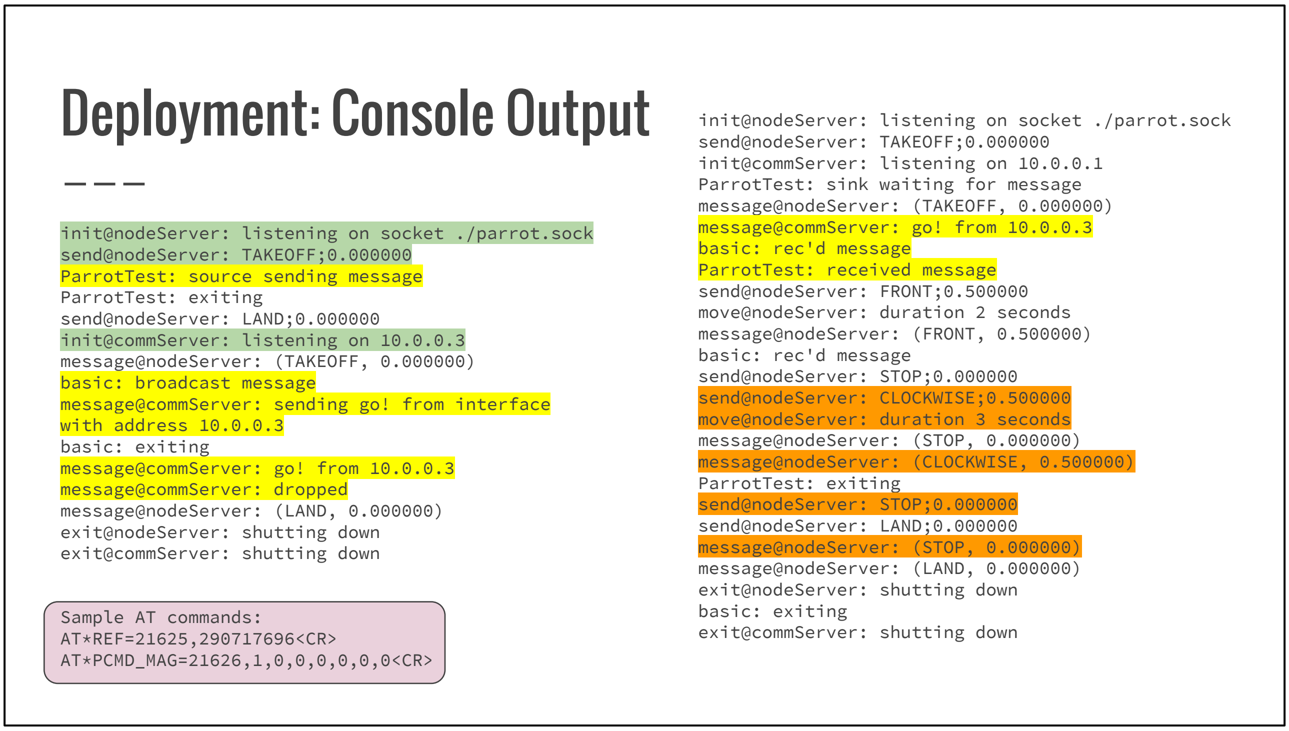
A comparison between the simulation environment as run on a single machine (as a conventional simulation), and deployed to actual quadcopters.

- The visualisation thread disappears, as there is no use for visualisation when you have the real thing
- Each host has its own sharded instance of the environment
- Each host has a communications server which listens for network traffic from other hosts
- Each drone has a node-js server which acts as an intermediate step between the simulator and the ar-drone middleware



Left hand side: the reference deployment setup as assembled in the lab. Make use of two Raspberry Pi Model As, with wifi adapters paired to the quadcopters. An off the shelf switch is used to connect them via ethernet to a laptop which acts as a base station.

Right hand side: the reference setup as deployed at the back of the Department of Computer Science, and a picture of one of the drones in flight (credit: David Richardson).



Deployment: Console Output

```
init@nodeServer: listening on socket ./parrot.sock
send@nodeServer: TAKEOFF;0.000000
ParrotTest: source sending message
ParrotTest: exiting
send@nodeServer: LAND;0.000000
init@commServer: listening on 10.0.0.3
message@nodeServer: (TAKEOFF, 0.000000)
basic: broadcast message
message@commServer: sending go! from interface
with address 10.0.0.3
basic: exiting
message@commServer: go! from 10.0.0.3
message@commServer: dropped
message@nodeServer: (LAND, 0.000000)
exit@nodeServer: shutting down
exit@commServer: shutting down
```

```
Sample AT commands:
AT*REF=21625,290717696<CR>
AT*PCMD_MAG=21626,1,0,0,0,0,0,0<CR>
```

```
init@nodeServer: listening on socket ./parrot.sock
send@nodeServer: TAKEOFF;0.000000
init@commServer: listening on 10.0.0.1
ParrotTest: sink waiting for message
message@nodeServer: (TAKEOFF, 0.000000)
message@commServer: go! from 10.0.0.3
basic: rec'd message
ParrotTest: received message
send@nodeServer: FRONT;0.500000
move@nodeServer: duration 2 seconds
message@nodeServer: (FRONT, 0.500000)
basic: rec'd message
send@nodeServer: STOP;0.000000
send@nodeServer: CLOCKWISE;0.500000
move@nodeServer: duration 3 seconds
message@nodeServer: (STOP, 0.000000)
message@nodeServer: (CLOCKWISE, 0.500000)
ParrotTest: exiting
send@nodeServer: STOP;0.000000
send@nodeServer: LAND;0.000000
message@nodeServer: (STOP, 0.000000)
message@nodeServer: (LAND, 0.000000)
exit@nodeServer: shutting down
basic: exiting
exit@commServer: shutting down
```

As you can see, even a simple example is made difficult to understand when considering 10 threads across two devices. All this program does is wait for an init message and then moves forward, turns, and lands.

Green, on startup:

- The node-js server, which talks to the middleware, which talks to the drone, starts up and listens on a unix socket
- The sharded environment instructs the drone to take off
- The communications server begins listening on the correct network interface

Yellow, sending a message:

- The program on the left hand side pushes a message to the communications module
- The communications module on the left broadcasts the message through the sharded environment instance
- The communications server on the left sends the message via multicast over a raw UDP socket to the node on the right
- The communications server on the left receives the message and drops it (it is the message origin)
- The communications server on the right receives the message from the multicast address
- The communications module on the right receives the message from the

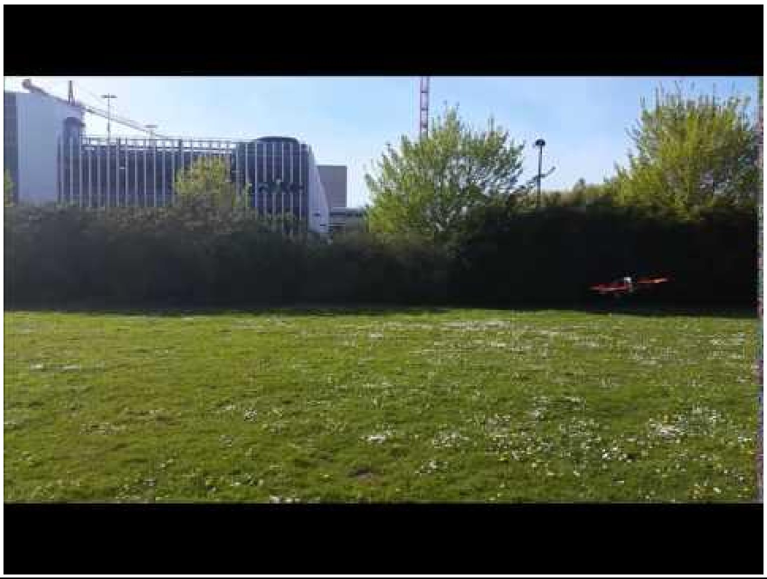
- communications server
- The program on the right receives the message from the communications module

Orange, moving:

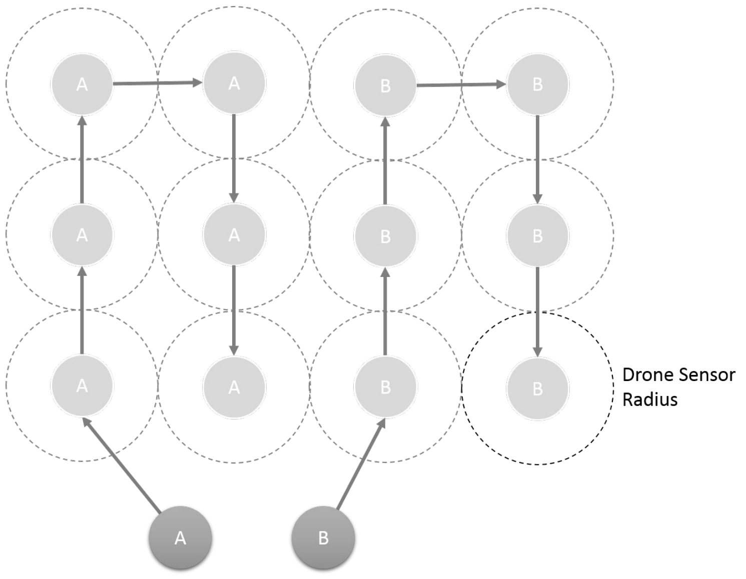
- The program calls the environment function to move, which calculates how long the drone must fly for, and sends a message to the node server over a unix socket
- The node server receives the message and sends the appropriate AT command to the drone
- After the right amount of time has passed (real time), the environment instance sends the stop command to the node server
- The node server receives this message and sends an AT command to the drone instructing it to hover

AT commands are the actual messages that are sent to the drone firmware. They are comprised of AT* followed by a command, then an equals sign and a number of arguments. The first command in the example is the takeoff/land instruction and the second for movement using the inbuilt magnetometer.

Collision Avoidance



Problem Division



Problem Division

Deployment: Problems

There were a number of issues which were encountered during the deployment process which we weren't able to model under lab conditions:

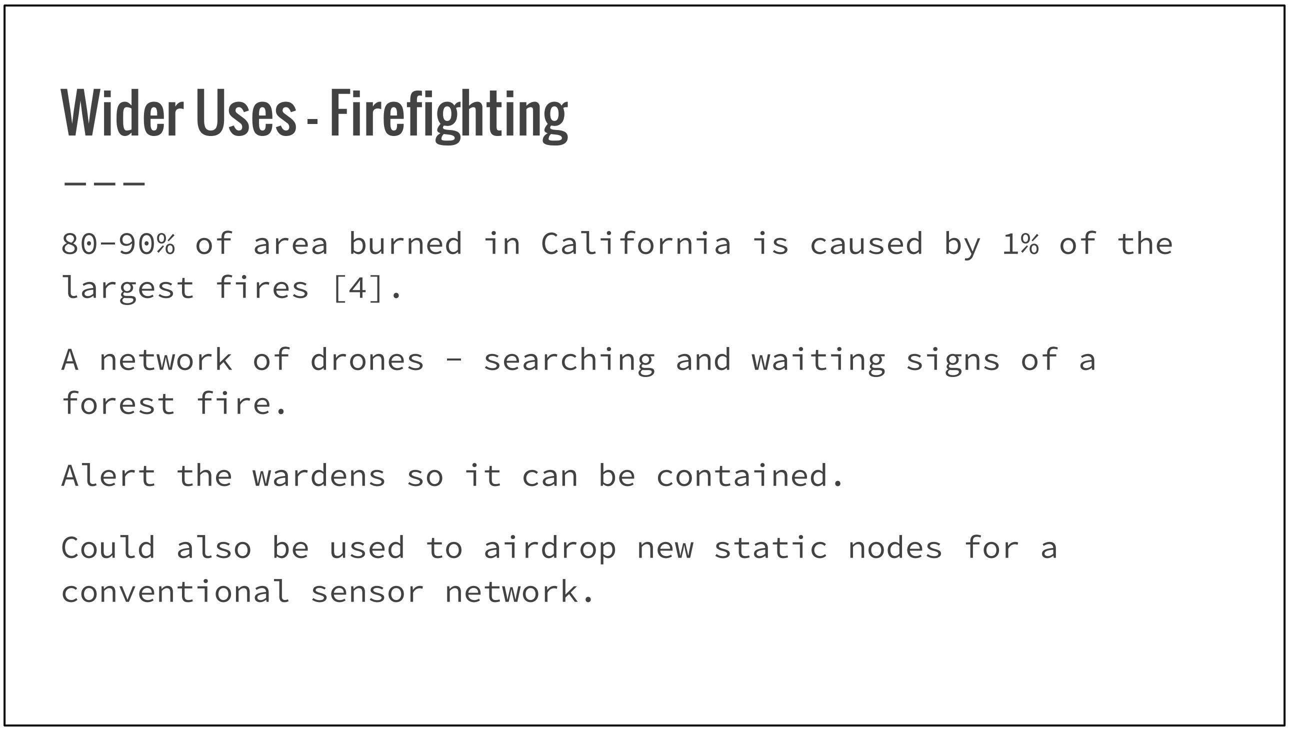
- The first takeoff failed because the front rotor of the drone got caught in the long grass. When the first command was given by the program the back rotors caused the drone to flip and cut out
- The second takeoff failed because the wind blew the drone off course. While the firmware attempted to stabilize and the program continued to execute as normal, the altitude was lower than expected, meaning the drone careers into the grass.
- The third takeoff failed because the control program crashed after issuing the forward command. As a result, the instructions that should have followed (most notably the one to hover) never reached the quadcopter and it sped off towards the Zeeman building.

We were able to learn from these disruptions though, and as a direct result of the latter run we made a script which would connect to and ground malfunctioning drones.

Wider Uses - Why Drones?

Asking the question “where are drones better than people?”

- Places where it is unsafe for humans.
- Situations where it is cheaper for unmanned work.
- When needing a view from the sky.



Wider Uses - Firefighting

80-90% of area burned in California is caused by 1% of the largest fires [4].

A network of drones - searching and waiting signs of a forest fire.

Alert the wardens so it can be contained.

Could also be used to airdrop new static nodes for a conventional sensor network.

Suitable using our system because:

- Basestation to collect data
- Can pass messages via nearby drones to talk across a large area.
- Can easily alter the functionality of the drones.

Wider Uses - Mapping

Many places too dangerous for people to go [5]:

- Abandoned Mines
- Unstable Buildings
- Areas of high radiation or toxicity
- Space…

Each drone maps a different area. Information passed back to base station for integration into a complete map.

Geographical or Topological.
Suitable for our system because:

- It is easy to disseminate areas between the drones
- Easy to pass information back to the basestation to combine into a map
- Can be useful to map from the sky

Outreach Implications

The 2015 RCUK Public Engagement with Research Strategy [2] sets out three main areas of research engagement:

- Involving the public with the research process
- Enabling public views to inform research strategies
- Engaging and inspiring the next generation

We believe that octoDrone is an excellent tool for academic outreach. We have created a platform which is powerful and extensible, whilst keeping it accessible to students.

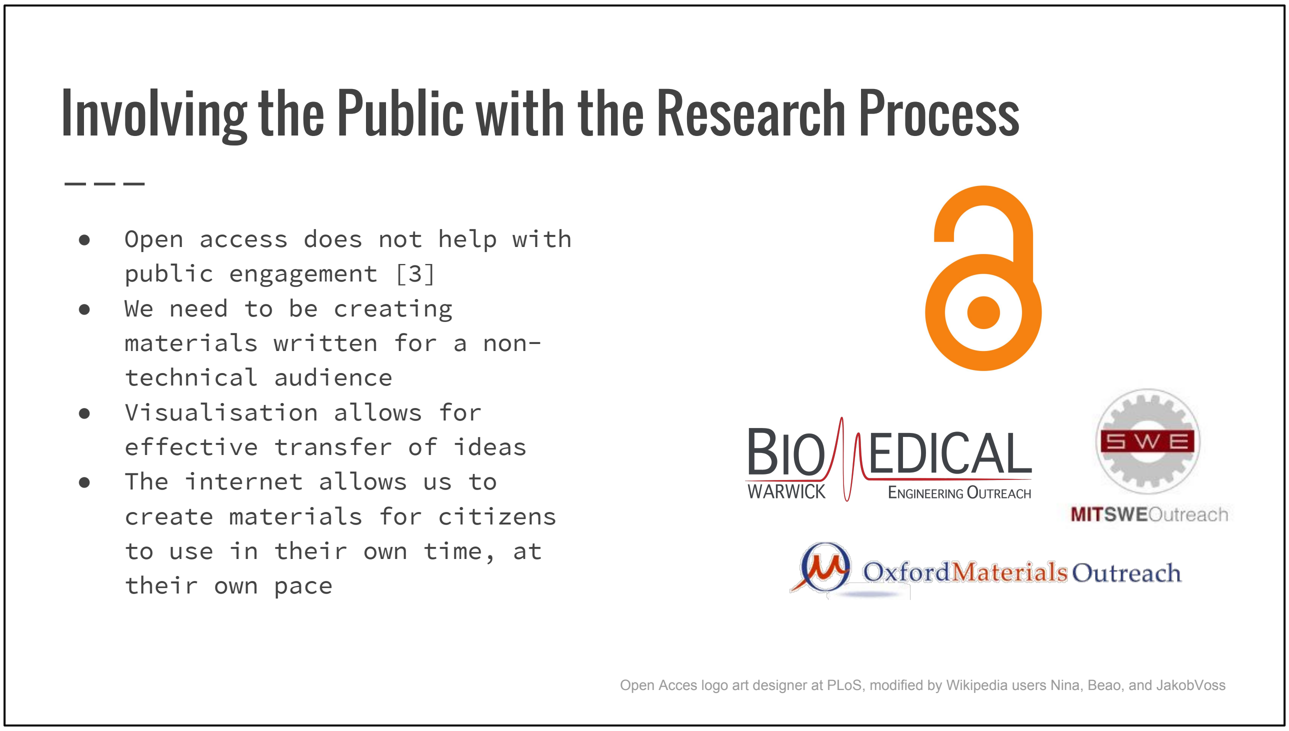
Before we go into more detail about specifics, a quick overview of outreach - what it mean, and what it's about. Outreach has increasingly become a focus of academic research. Since 2007, research councils in the UK have made a concerted effort to increase recognition and funding for outreach events.

The three main ways this manifests itself is in engaging the public - making them aware of the research that is taking place, as well as encouraging social participation.

It is important that this is a two way process though, that the views and attitudes in modern society shape the research that we carry out.

Finally, we need to be engaging the next generation. In order to ensure a supply of future academics and researchers we need to be able to demonstrate our work. Beyond this, a society with more informed citizens is beneficial for everyone.

With that in mind, I would like to take the time to evaluate our project as a tool for outreach. We have already had members of the department express an interest in using octoDrone to engage with students, and the hope is that they will be able to use our work to further their own outreach goals.



Involving the Public with the Research Process

- Open access does not help with public engagement [3]
- We need to be creating materials written for a non-technical audience
- Visualisation allows for effective transfer of ideas
- The internet allows us to create materials for citizens to use in their own time, at their own pace

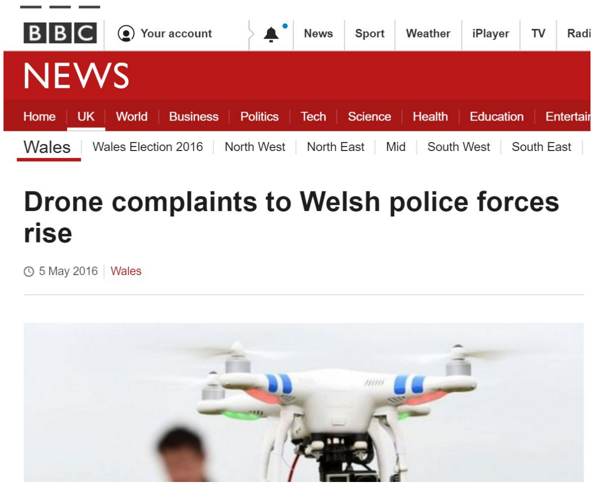
Open Acces logo art designer at PLoS, modified by Wikipedia users Nina, Beao, and JakobVoss

When considering how to make research accessible, open access is commonly mentioned. But while the move to make more research open access is undoubtedly a positive one, it does not help with public engagement. No matter how easy it might be to obtain access to a paper, there will only ever be a very small minority of the population who know of its existence, and even fewer who will understand it, because they were written for a different audience.

So what we are considering here is the act of explaining research for the public. Particularly with mobile sensor networks, it can be hard to explain the operation of large systems with many layered mechanics. Often visualisation is the best way to communicate the workings of these constructs, and our simulator does just that. It presents a visual, easy to interpret, and repeatable way for those outside the academic community to engage with the work that we carry out. Compared to showing a paper or an RFC, showing an interactive representation is a very effective way of getting a message across.

But outreach is about more than academics demonstrating their work. The internet means that we have an opportunity to make tools such as ours available to all online, allowing interested citizens to learn at their own convenience and pace. Distributing a tool such as octoDrone alongside an online series such as Crash Course could be a big step to getting people involved.

Enabling Public Views to Inform Research Strategies



Drone complaints to Welsh police forces rise

5 May 2016 | Wales

BBC (2016) Drone complaints to Welsh police forces rise. Published 05/05/2016.

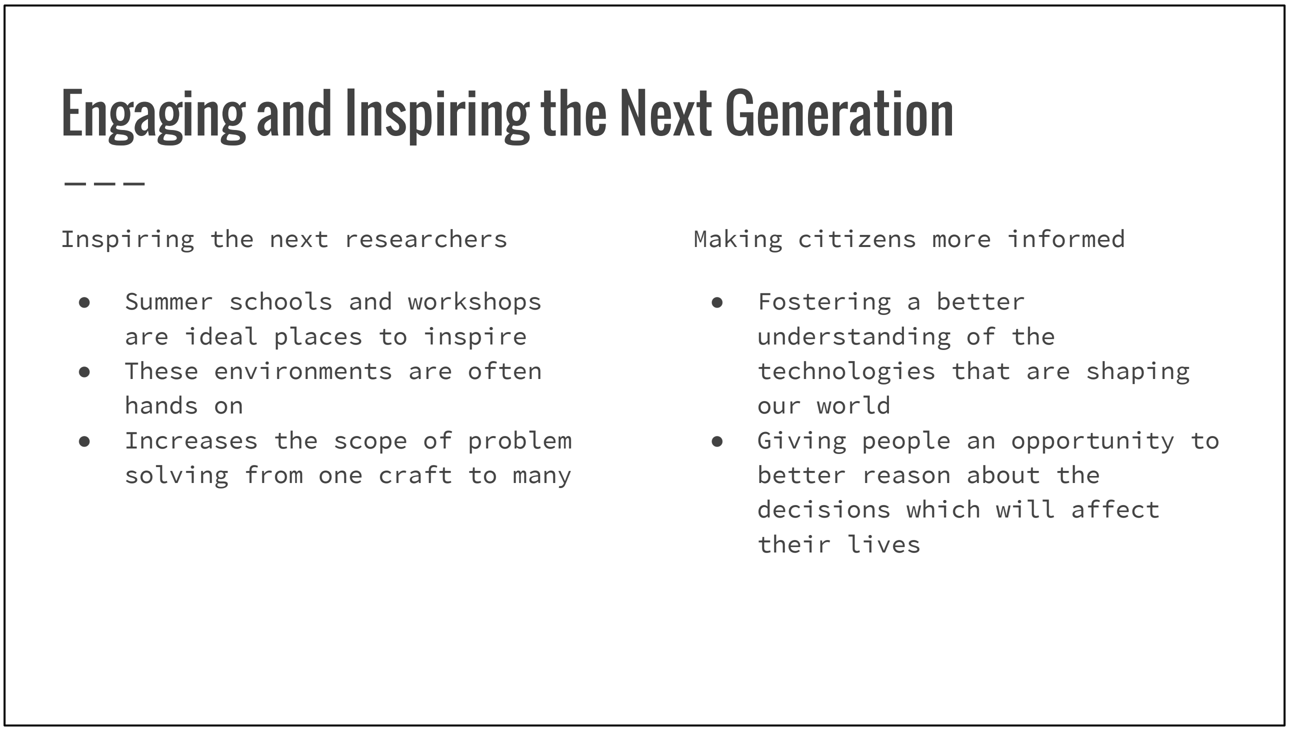
- Research strategies should be driven by the issues facing society
- The public need to know about a topic in order to have an opinion about it [3]
- This can primarily be achieved through outreach tools such as ours
- Familiarity helps to counteract negative media presentation

So far we have only considered one way communication of academic ideas, but in reality it is important to listen to feedback and ideas from the public. In order to be relevant, what we spend our time and money investigating needs to be driven by our culture.

This is not to say that the public or government should have fine grain control over what we do as academics -at the end of the day it is academics who are best placed to know which areas need research- but that in order to be most effective our research goals should align with the issues that we face as a society.

How does this relate to our project? We've seen how important outreach is in making others aware of the research that's taking place so that they can offer their own views and inputs, but they have to know about it first. This is strongly linked with the previous point in that we see octoDrone as a tool to make people more aware of this active topic of academic research.

As drones become more and more mainstream we'll start to see this happen anyway, but the media has a habit of only presenting the negative aspects of new technologies, which dampens public opinion.



Engaging and Inspiring the Next Generation

Inspiring the next researchers

- Summer schools and workshops are ideal places to inspire
- These environments are often hands on
- Increases the scope of problem solving from one craft to many

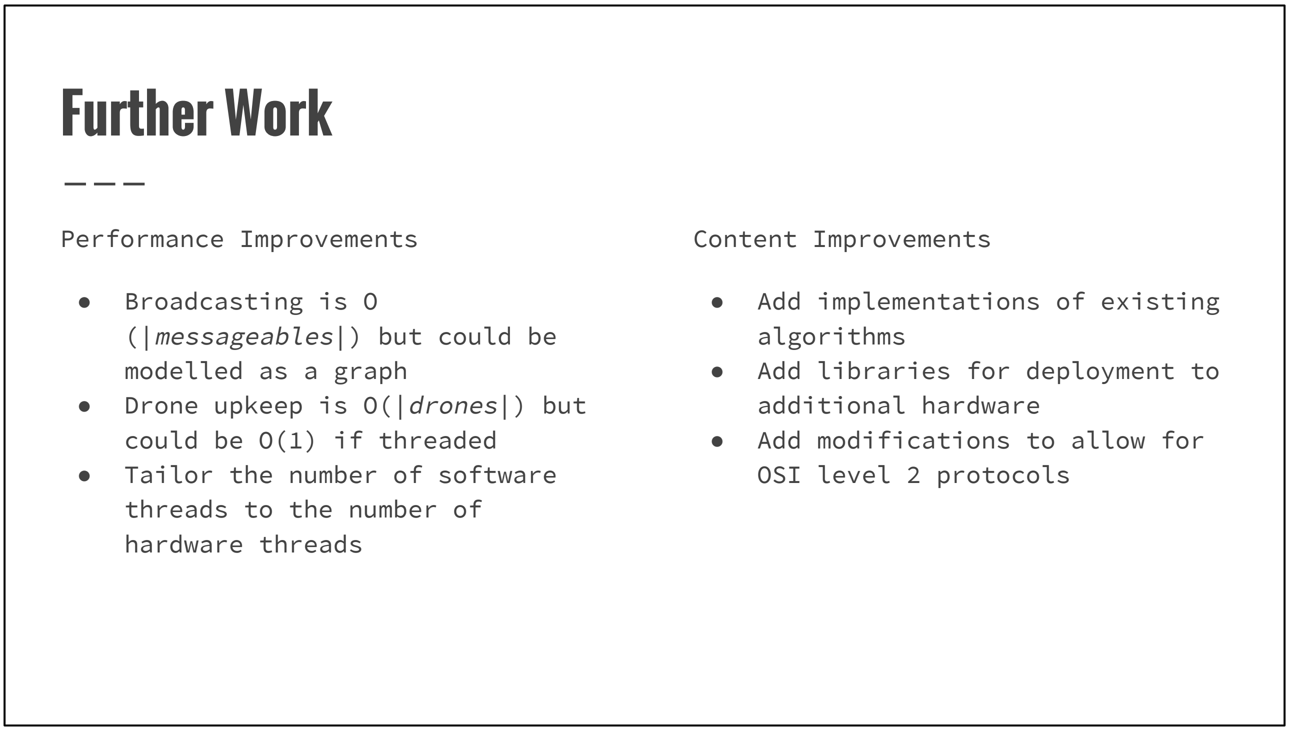
Making citizens more informed

- Fostering a better understanding of the technologies that are shaping our world
- Giving people an opportunity to better reason about the decisions which will affect their lives

But most people do not own a quadcopter or have access to one - a major barrier to entry.

One of the best ways for students to get hands on experience with drones is currently summer schools like the headstart residential that Warwick hosted last summer. It's through sessions at events like these that we can start to teach students how to solve problems with quadcopters. But hands on time is expensive and inefficient, especially if there's a learning curve involved. We can solve this with octoDrone by preparing participants with the theory before they have the opportunity to use the real thing. In addition to this, simulation leads to thinking in terms of networks instead of individual units. This is important because many of the most influential and exciting uses of quadcopters is with swarms.

To close out this section of the presentation I thought it would be useful to mention one of the oft-forgotten and most hidden aspects of outreach - to make people more informed. The technology we've been discussing today will shape the lives of everyone in society and by creating a tool which can be useful in this regard we hope that we might be able to make better decisions as a community about how quadcopters are used and legislated for.



Further Work

Performance Improvements

- Broadcasting is O(|*messageables*|) but could be modelled as a graph
- Drone upkeep is O(|*drones*|) but could be O(1) if threaded
- Tailor the number of software threads to the number of hardware threads

Content Improvements

- Add implementations of existing algorithms
- Add libraries for deployment to additional hardware
- Add modifications to allow for OSI level 2 protocols

When it comes to considering further work, opportunities tend to fall into two categories:

Improving the performance of the simulator by introducing optimisations, or
Adding content to the simulator that would be useful for modelling sensor networks

In terms of the former, there are a number of areas in which the performance of the simulator leaves something to be desired. When broadcasting a message, each messageable is checked to see if it is in range. In fact, most of the time we know enough about the topology of a simulated network to be able to partition the simulation space in a way which means that we do not have to exhaustively check every node. By considering this as a reachability problem, one could construct a graph of the simulation elements, with edges between those nodes which are in range of each other. This approach would, however, require some notion of a maximum range employed in a simulation, which would have to be provided by the user.

In terms of parallelism, the upkeep processing for drones presents another opportunity to improve the efficiency of the simulator. Upkeep methods are independent of one another (required by the fact that there is no deterministic order in which they are performed) which means that they can be executed at the same time without violating the correctness of a simulation. The gains made by this optimisation would scale with the amount of processing required by drones every simulation tick. More processing means more efficiency, with simple solutions experiencing little to no net benefit from parallelisation.

Finally, it is worth investigating adding the ability to change the number of threads created by octoDrone. For hardware which is capable of running many threads in parallel, such as the DCS lab machines, the most performant solution has a large number of threads. With low end hardware, however, such as Raspberry Pis or advanced Parrot quadcopters, this is not the case. Here there is a performance penalty incurred by repeatedly switching between the many threads used by the simulator.

In terms of improvements to the content available for octoDrone, the answer to the question of further work is inevitably “more”. Be this communications algorithms or deployment libraries, the main obstacle for octoDrone being useful to research communities is the lack of add ons available. This is something that can only really be changed with a combination of time and either an active development community or full time programmers. The simulator is flexible enough that most tasks can be performed by it with relatively little modification, the most notable one here being simulation of hardware collisions to allow for use ofOSI level 2 MAC protocols.

References

[1] John Paulin Hansen, Alexandre Alapetite, I. Scott MacKenzie, and Emilie Møllenbach. The use of gaze to control drones. In Proceedings of the Symposium on Eye Tracking Research and Applications, ETRA ’14, pages 27–34, New York, NY, USA, 2014. ACM.

[2] Research Councils UK, (n.d.). RCUK Public Engagement with Research Strategy. [online] Available at: http://www.rcuk.ac.uk/documents/publications/rcukperstrategy-pdf/ [Accessed 4 May 2016].

[3] Hone, D. (2013). Why should academics get involved in outreach?. The Guardian. [online] Available at: https://www.theguardian.com/science/lost-worlds/2013/jan/02/dinosaurs-fossils [Accessed 4 May 2016].

[4] Stephen J Pyne et al. *Fire in America. A cultural history of wildland and rural fire.* Princeton University Press, 1982

[5] Gem Systems Advanced Magnetometers. Uavs - pathway to the future - gem systems, 2015

Questions