Australian National University

SPACE CONVOY CONTROL

Assignment for all students of Real-time and Embedded Systems

This is the only marked assignment of the course. Thus extra care is expected on all levels.

Overview

Regulating airspace is a classical and demanding real-time task and we will take it to the next level. Instead of handling glide slopes which are radially distributed around a static airport, we deal in approach paths from any direction towards a mobile charger grid – a task which can be embedded into a real-time approach schedule.

Your part is the design of the charge/control modules which go into each vehicle.

A swarm of vehicles will be provided, which have default behaviours to keep them in motion, together and collision free.

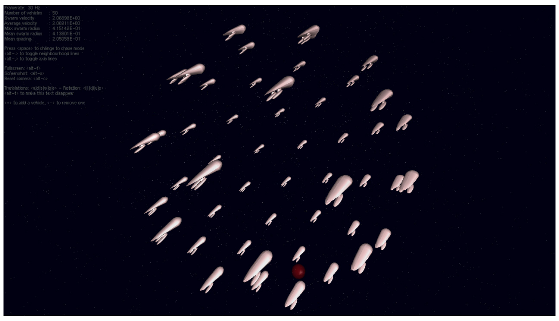
All vehicles have a local "charge" which need to be replenished by passing through "energy globes". For unknown physical reasons only passing of three energy globes in fast succession replenishes the charge to full (see small graphic on page 2). Vehicles which run out of charge mysteriously disappear.

As this is a science fiction assignment, we can safely assume that we have semiconducting coils in on-chip, miniature fusion reactors which supply the comms and on-board computers. Hence all energy loss will be due to propulsion, which means that the loss rate is proportional to the current acceleration and thus acceleration-free, "drifting" vehicles are not consuming energy.

Sensors, Actuators & Communication

Each vehicles runs a dedicated task together with an interface to the local sensors, communication interfaces and actuators.

The sensors include position, velocity and acceleration as well as current charge. If the vehicle is close enough to one or multiple energy globes to utilize them, the sensors also display the position and current velocity of all close energy globes.

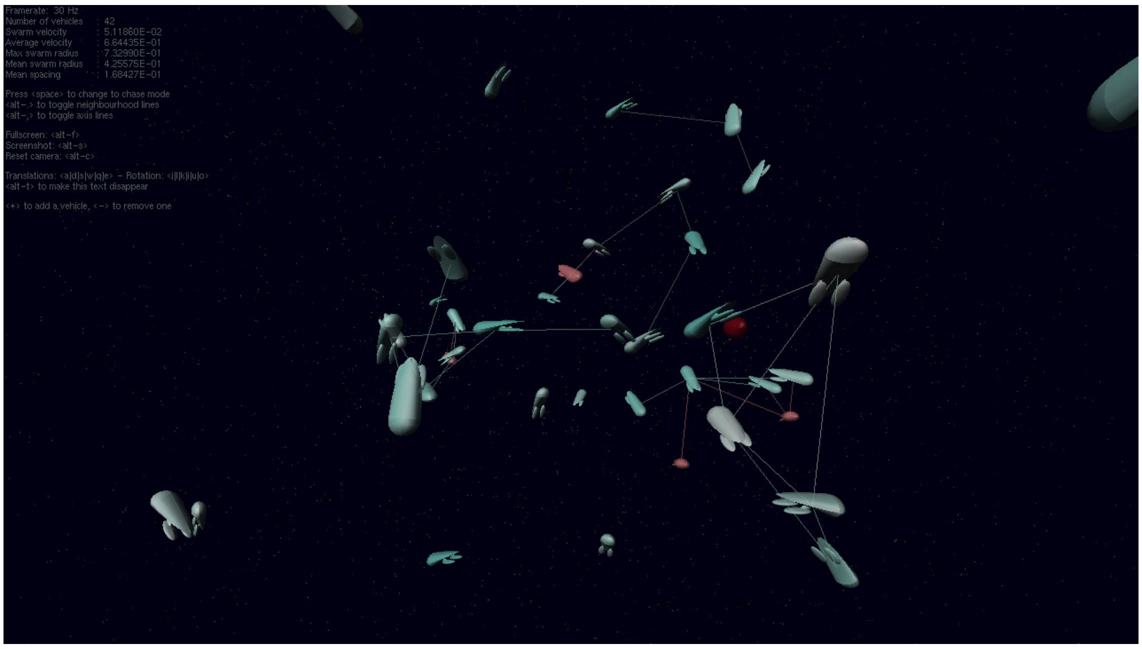


```
Framerate:  30 Hz
Number of vehicles      :  50
Swarm velocity          :  2.06899E+00
Average velocity        :  2.06911E+00
Max swarm radius        :  4.15142E-01
Mean swarm radius       :  4.13801E-01
Mean spacing            :  2.05059E-01

Press <space> to change to chase mode
<alt-.> to toggle neighbourhood lines
<alt-,> to toggle axis lines

Fullscreen: <alt-f>
Screenshot: <alt-s>
Reset camera: <alt-c>

Translations: <a|d|s|w|q|e> - Rotation: <j|l|k|i|u|o>
<alt-t> to make this text disappear

<+> to add a vehicle, <-> to remove one
```

The actuator system consists of setting an absolute destination position and throttle value. The underlying cruise control systems automatically handles the steering and acceleration. Once the destination point has been reached the throttle automatically jumps back to idle which means that default swarming behaviour takes over. The vehicles do not slow down when approaching the destination point and rather pass through the destination point. This helps to keep the controls fluent and the vehicles in motion. Collision avoidance reflexes are always active and prevent vehicles from crashing into each other. Note that destinations might become unreachable, if multiple vehicles are bound for the same destination.

The vehicles are also equipped with a message passing system which allows to broadcast a message which will be received by all vehicles on close proximity.

Finally there is also a function which allows direct access to the underlying, secret clock of the world. `Wait_For_Next_Physics_Update` will put the task to sleep until anything actually happened (which includes communication). This relieves the vehicles from busy waiting on the world to change.



```
Framerate:  30 Hz
Number of vehicles      :  42
Swarm velocity          :  5.11860E-02
Average velocity        :  6.64435E-01
Max swarm radius        :  7.32990E-01
Mean swarm radius       :  4.25575E-01
Mean spacing            :  1.68427E-01

Press <space> to change to chase mode
<alt-.> to toggle neighbourhood lines
<alt-,> to toggle axis lines

Fullscreen: <alt-f>
Screenshot: <alt-s>
Reset camera: <alt-c>

Translations: <a|d|s|w|q|e> - Rotation: <j|l|k|i|u|o>
<alt-t> to make this text disappear

<+> to add a vehicle, <-> to remove one
```

The Animation

The provided graphical animation of the swarm offers third person views as well as the view from one of the vehicles while it is passing through the swarm. The communication range can be visualized by drawing connecting lines between all vehicles which are currently in communication range. The colours represent their charging state as well as the control state. Turquoise vehicles are currently following their swarming instincts are not explicitly controlled by the associated task. The colour saturation reflects the level of charge. Once vehicles go into manual control (throttle and destination is set) they turn to a more red colour schema. The energy globe(s) are dark ruby coloured spheres.

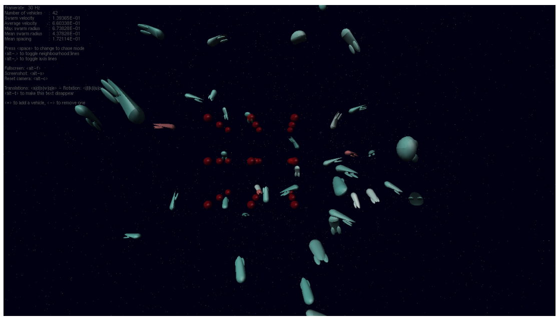
Design Constraints

The final solution which is requested from you should be deployable on actual vehicles. While the provided interfaces make it easy to guarantee that, there is no constraint which would force you to use them. “Looking underneath the hood” is encouraged of course if you want to see how such a simulation can be implemented on a computer system. You can, in fact, ignore any provided code framework and can come up with your own or an adapted version of the provided framework – as long as you can argue that the control systems would still be deployable on individual vehicles moving under physically realistic constraints and with limited communication ranges. Also: nothing stops you from enhancing to existing framework to cater for more real-world constraints, like lossy communication channels or imperfect sensing. Nevertheless the first stage of the assignment will allow you to introduce additional means of communications which are not necessarily physically implementable.

Real-time Constraints

All calculations inside the vehicle tasks have an implicit deadline given by the update from the underlying physics engine. This deadline is not hard as seen by the local tasks, yet all tasks overrunning this deadline in synchrony will slow down the simulator. Simulated time is not affected by this – only the update time intervals will become larger.

In order to understand the timing and synchronization constraints of your individual vehicle tasks, you are encouraged to analyse the potential blocking interactions, starting from the vehicle interface package.



Design Goals

The overall design goal is to coordinate the swarm to a degree that its behaviour becomes predictable in the sense that all vehicles are guaranteed to be recharged on a regular basis. Traffic shall be coordinated in the sense that collisions (or the potential thereof) are to be avoided by introducing appropriate scheduling and communication. The task can be solved in two stages:

a. Allowing a central coordinator.
b. Fully distributed.

The first stage still allows for a central coordinator to be introduced and all tasks are allowed to communicate with this entity (or multiple thereof). The implementation of those central instances can employ shared memory as well as message based forms of communication. Some are obviously questionable to impossible in a physical deployment of your system, yet this stage might help you to develop ideas which can then be considered for the second stage.

The second stage does not allow for a central coordinator and all planning and scheduling now needs to be done on the individual vehicles only using local communication. This is hard.

Energy Globe Configurations

Two pre-programmed energy globe configurations are available. One where the a grid of 27 globes stays roughly in the middle of the swarm at all times, and one where the a wider grid of again 27 globes will glide through space in a linear motion. You can change the configurations by changing the according constant in the Configuration package. Feel free to add or test others – the only option which is discourage as too unrealistic is a stationary set of energy globes.

Potential Issues

The task can be a impossible to solve, if no vehicle found an energy globe before the initial charges run out. Don't worry about this case – this is just space travellers' bad luck.

The Programming Framework

The provided code has been successfully compiled and tested on:

- **Linux**: Ubuntu version 12.04, and lab computers. Depending on the install version, it might be necessary to install glutg3 and glutg3-dev via the built-in package manager.
- **Mac OS**: running under OS X version 10.6 and 10.8.
- **Windows**: XP, Win7 and Win8. The freeglut library need to be in the same directory as the executable (already placed to the right spot in the provided project).

On the top level directory of the code framework you will find three different project files (one for each major platform). Open the one fitting the computer you are currently sitting in front of in GPS (GNAT Programming Studio) and execute "Build all". This should produce an executable binary file for your platform. Each project file also has two build configurations which can be selected within GPS: *Development* and *Production*. The first will include all run-time code checks, while the second will optimise the code for maximal performance. While the latter is irrelevant in terms of a real-time project, it might still allow you to run tests with a larger number of vehicles. Experienced performances are also dependent on your installed graphics processor.

Parts of the provided framework are based on the Globe_3D project (an OpenGL 3D engine), which is maintained by Gautier de Montmollin.

Deliverables

You need to submit a report (in pdf) as well as your code. If you convince us about a great design and that you understood the problem to a degree that you would not need to implement and run tests, then you don't need to provide code. In this case you need to be prepared to explain your proposed implementation in the exam in great detail.

Report

Your report should primarily answer the following questions:

- How can you guarantee that a minimal number of vehicles will sustainably survive?
- What assumptions did you make to provide this guarantee?
- Which parts of your system need to/could be precalculated and which parts need to/could be experimented? How did you decide in cases where both was possible?

- How did you calculate or measure the maximal number of vehicles which you can sustain?

Some guidelines to structure your report:

- Based on the general description of the problem at hand, make the constraints which you have addressed in your design explicit and list them as precisely as you see fit.
- Documentation of your design. Specific emphasis should be given to explain your design decisions. Give reasons for each of those. Make clear which constraints you employed as 'driving concepts', which have been considered, and which have been purposefully ignored (for instance to allow for a cleaner, easier maintainable design).
- Provide documentation of test runs. Give a precise motivation for each of your tests.

The following questions might help you to evaluate your design:

- How does your design scale?
- Which real-time constraints are or could become critical if the system is extended?
- Do you provide for graceful degradation in case that parts of your system become unresponsive or provide 'unreasonable' information?
- To which parts of your system could you apply strict algebraic verification?

Code

Submit the manipulated packages (vehicle task & message structure) together with the packages which you added on top. Your code will be evaluated according to common professional practice. We do not enforce a specific coding schema, but request consistency and a general high standard on the basic coding level. Make sure all your identifiers have good names, all scopes and access constraints are set as tight as possibly, and full use has been made of compile time checks.

Expect to be limited to distinction range marks if your design only considers stage one solutions and to find yourself in "HD" range if your design is a convincing solution for stage 2. Yet, those are only guidelines, and an outstanding stage one solution can be considered for full marks as well. Allow yourself plenty of time to come up with a solid concept first. Without a clear idea you are bound for chaos in this assignment.

General

Use graphical or any other means to structure and express your ideas as precisely as you can. Be also prepared to 'defend' your documentation in an oral exam situation. The documentation should be printed and submitted at least three days before the laboratory exams. Overall assignment time is six weeks. Due date according to web-site.