# A Beginner's Guide to the Truckworld Simulator

Steve Hanks, Dat Nguyen, Chris Thomas
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Technical Report 93–06–09

# A Beginner's Guide to the Truckworld Simulator

Steve Hanks, Dat Nguyen, Chris Thomas
Department of CS&E, FR–35
University of Washington
Seattle WA 98195

June 25, 1993

UW–CSE–TR 93–06–09

**Abstract**

The Truckworld is a multi-agent planning testbed that features uncertainty, exogenous change, complex interaction among objects in the world, semi-realistic sensing, and facilities for communication and other interaction among agents.

This document is designed to give the first-time user a feel for the testbed's capabilities, and also will provide instructions on how to obtain the code and run a standard configuration of the simulator.

## Contents

## 1 Simulator Overview

The Truckworld is a multi-agent planning testbed that features uncertainty, exogenous change, complex interaction among objects in the world, semi-realistic sensing, and the capacity for communication and other interaction among agents.

This paper should give the reader of sense of the simulator's structure and an agent's capabilities. We will ignore many of the simulator's features, in particular deferring a discussion of multiple agents and communication to a subsequent paper [2]. A reference manual [1] provides still more detail.

Figure 1 displays a simulation in progress; each truck agent has its own display window. A display consists of three parts:

- A panel showing the truck's status: its fuel level, heading, speed setting, operational status, and so on. This panel appears in the window's upper-left corner.
- A panel showing what objects are present at the truck's current location. This panel appears at the far right. Notice that the truck's current location is a place called SENSOR-NODE and in fact the truck itself is pictured at this location along with some other objects.
- A panel showing the world map. Notice the location SENSOR-NODE is highlighted, indicating the current location of the truck. The lines connecting the locations indicate connecting roads: there is a road that leaves BATTLE in the southeast direction, for example, and arrives at a location named DEMAND-2 coming from the northwest.

You make things happen in the Truckworld by issuing commands to the truck. All changes in the Truckworld are triggered by *events*. Commands to the truck generate events; there can also be *exogenous events* like rainstorms that are not generated by any truck, but we will ignore these for now.

Below we will describe the truck's command set—what commands it responds to and what changes they cause in the world—but first we describe the simulator's spatial model.

### 1.1 Space and geometry

The Truckworld has a very simple model of space, built around the concept of *containment*. Think of the entire world as a big container, containing all the locations pictured on the map and all the roads. Each location is also a container; you can see from Figure 1 what the SENSOR-NODE location currently contains.

The truck is contained in SENSOR-NODE, but it is also a container, containing two arms, two cargo bays, a fuel tank, and a tire bay, among other things. The truck's cargo bays and its arms are themselves containers, though they are presently empty.

Other objects in the world can be containers too—you can create boxes that contain other objects.

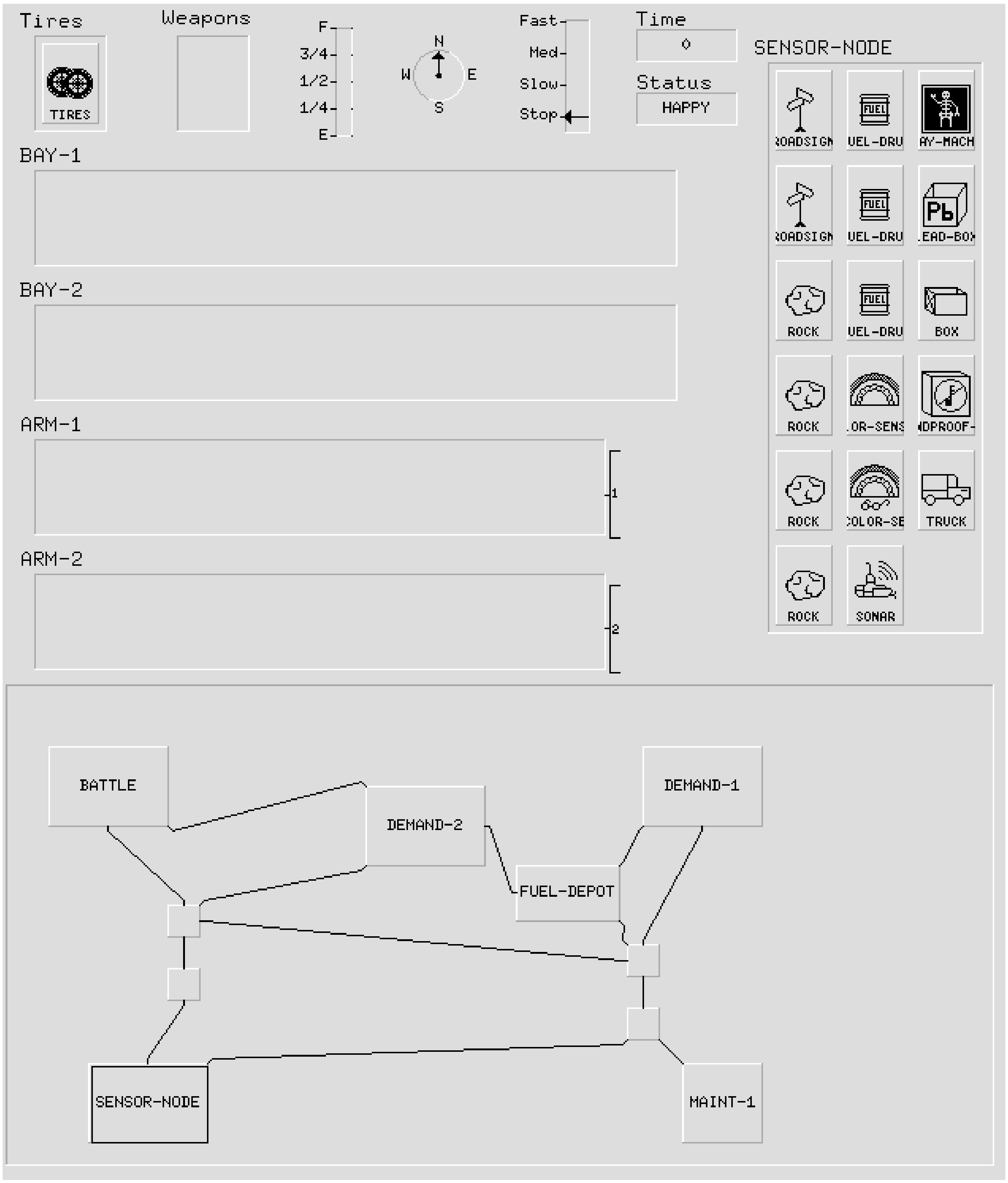

Figure 1: A Truckworld Display

The only spatial information we maintain about an object is its current *container*, and its *position* within that container. Positions are numbers starting with 0, and each object, regardless of its size, occupies one position in its container. In the display for SENSOR-NODE you see a "road sign" at position 0, another road sign at position 1, a rock at position 2, and so on. The truck itself occupies position 16. Containers usually display their contents from top to bottom, and from left to right.

*Vessels* are another type of container, but they hold fluids instead of solid objects—the most important kind of vessel is the *fuel drum*, which holds gasoline. Fluid from one vessel can be poured into another vessel, mixing them together in the second.

The connectivity of the roads and locations is the only other spatial concept in the Truckworld. The roads and locations form a graph, and travel in the Truckworld consists of traversing this graph: if a truck wants to move from SENSOR-NODE to the node north of it, it sets its heading to `NORTH`, sets its speed to something other than `STOP`, and issues a single command `truck-move`. If all goes well it ends up at the location at the end of the road to its north. Navigation in the Truckworld is simple in the sense that a single command moves the truck, but many things can affect how the `move` turns out:

- without enough gas the truck ends up stuck on the road,
- moving too fast on a winding or icy road can cause the truck to roll,
- certain roads have weight limits and exceeding them leads to disaster, and so on.

## 2 The Truck

We already mentioned the truck's main components: its arms, its cargo bays, its fuel tank. The section discusses the commands a truck can execute. Detailed syntax for these commands appears in Section 5.

The truck can do four things: manipulate its arms, set its internal state, move from place to place, and manipulate objects.

### 2.1 Arm commands

Arm manipulations are of two sorts: moving the arms around, and grasping objects with them. There are three basic "places" the arm can be: folded, inside the truck, or outside the truck. In the last case the arm is considered to be inside the same container (map location) that currently contains the truck itself. The arms start out folded, have to be folded while the truck is moving, and to go from inside to outside must be folded in between.

Here are three basic arm-moving commands. In each case `<arm-name>` refers to the internal name for the arm; the arm's name also appears on the truck's display panel. The truck in Figure 1 has two arms, named `ARM-1` and `ARM-2`.

- `(arm-move <arm-name> inside)`
  Arm must be folded. Move the arm inside the truck.
- `(arm-move <arm-name> outside)`
  Arm must be folded. Move the arm outside the truck.
- `(arm-move <arm-name> folded)`
  Fold the arm.

Once inside the truck the arm can move to some distinguished locations:

- `(arm-move <arm-name> <internal-location>)`

where `<internal-location>` can be `tire-bay`, `weapon-bay`, `fuel-tank`, or `<bay-name>`. Valid bay names for the truck in Figure 1 are `BAY-1` or `BAY-2`. Each of these objects is itself a container, except for `fuel-tank`, which is a vessel.

When the arm is outside the truck, or more generally at any container, the basic command for moving an arm is

- `(arm-move <arm-name> <position>)`

where `<position>` is a non-negative integer that is a valid position number for the container currently containing the arm. The `inside` and `outside` forms actually apply to any container: if there's a box at position `n` of the truck's current location, the arm can first move `outside` (to the location), then to position `n`, then `inside` (the box), then can move to positions within the box, then `outside` puts it back outside the box at position `n` within the truck's current location.

Once an arm is at a particular location there are three things it can do: grasp something, ungrasp something, and pour something. The basic grasp command is

- `(arm-grasp <arm-name>)`

which grasps the thing at the arm's current position. The arm has to have enough *capacity* to grasp the object. All objects have a mass-like attribute called *bigness*, and every arm has a limit to how much bigness it can hold. This restriction prevents the arm from picking up the truck, or fixed objects like roadsigns, for example. When an object is grasped it disappears from its old location and appears in the corresponding arm bay.

The basic ungrasp command is

- `(arm-ungrasp <arm-name> <position>)`

where `<position>` refers to the position *in the arm's bay* containing the object to be ungrasped. The object is deposited at the arm's current location, which must be empty when the ungrasp command is executed.

The *pour* operation transfers liquid from one vessel to another:

- `(arm-pour <arm-name> <position>)`

The arm must be holding a vessel at `<position>`, and the arm's current position must also contain a vessel. The `pour` transfers liquid from the held vessel to the one where the arm is located, until either the source vessel is empty or the destination vessel is full. This command is often used to fuel up the truck, transferring fluid to the truck's fuel tank. Here is a sequence of commands that would transfer some gasoline to the truck, given a world in the state displayed by Figure 1:

```
(arm-move arm-1 outside)
(arm-move arm-1 4)
(arm-grasp arm-1)
(arm-move arm-1 folded)
(arm-move arm-1 inside)
(arm-move arm-1 fuel-tank)
(arm-pour arm-1 0)
```

(The last position index is 0 because the arm is currently holding the fuel drum at position 0 within its arm bay.)

## 2.2 Truck commands

Commands that change the truck include those that alter its *controllers* and the one that causes it to move. A truck's controllers govern its speed and heading. Changing these values won't cause the truck to move immediately; controller values govern what *will* happen when the next `move` command is issued.

- `(change-heading <direction>)`
  Where `<direction>` is a compass point: `N`, `NW`, `W`, `SW`, `S`, `SE`, `E`, `NE`.

- `(change-speed <speed>)`
  Where `<speed>` is either `stop`, `slow`, `medium`, or `fast`.

- `(truck-move)`
  Send the truck on a journey. The arms have to be folded, the heading controller must be pointing in the direction of a road, the speed controller must be something other than `stop`, and there must be sufficient fuel.

Trying to set the speed or heading to a bogus value results only in a syntax error; the consequences for trying to move incorrectly are more dire. Moving in a direction where there is no road causes the truck to roll, and there's no escape unless you're lucky enough to be carrying a winch.[1] Moving without enough fuel will cause the truck to get stuck on the road, out of gas. You can refuel if you have fuel drums. Moving too fast gets to the destination quicker, but uses more fuel, and increases the likelihood of a mishap.

## 2.3 Manipulating objects

Many other objects, active and passive, can occupy the world, and the truck can affect them in two ways. The first is just by picking them up and moving them from place to place. The second is by a generic `set` operation, the general form of which is

- `(arm-set <arm-name> <position-or-NIL> &REST <arguments>)`

The second argument refers to the position in the arm bay of an object the arm is currently holding, or the object at the arm's current location (if `NIL`). The effect of additional arguments depends on the object being manipulated.

The effect of `set`ting an object depends entirely on the object. Some examples are:

- Setting a camera object might cause it to take a picture, which would later be "read" (see Section 3 below).
- Setting a lamp object to `on` might cause the lamp to be illuminated, making the truck's visual sensors more effective.
- Setting a widget-manufacturing machine might cause it to produce a widget.
- Setting a radio object to a string might send that string to whatever agent is holding the matching radio object.
- Setting a bomb might cause an explosion and break all the objects in the same container.
- Setting a rock probably won't do much.

That's the extent of the truck commands, except for the `read` command discussed below. Section 5 summarizes the commands.

The discussion to this point has oversimplified the effects of performing the various actions. A truck has various parameters that control the effects of its actions. Each arm has a "clumsiness" parameter, for example, that dictates the percentage of time a grasp operation fails. Roads have properties like *distance*, *drag* and *danger* that dictate how much time a `move` operation takes, how

[1] But see Section 4.2 below on how to cheat your way out of trouble.

much fuel it consumes, and how likely a mishap is. Roads also have a *composition* parameter—bumpy roads tend to break fragile objects in the truck's cargo bays, muddy roads tend to make the truck stuck unless it's equipped with mud tires. [1] contains a complete description of truck and world parameters, and their effects on truck commands.

## 3 Sensing

The truck gathers its information about the world through `sensor` objects. Sensor objects are like any others in that they have a *bigness*, occupy a position in a container, and can be grasped. The truck can manipulate sensor objects just like other kinds of objects. Sensors are special in that when they are activated (using `arm-set` or `truck-set`) they gather information about the world, which the truck can then retrieve using the `arm-read` or `truck-read` commands. The nature of the information depends on the sensor: a camera sensor captures information like color and shape, a scale sensor notes an object's bigness. Defining a sensor involves specifying what objects are accessible to the sensor, what properties of those objects should be noted, and how accurately the sensor should report those properties.

Different sensors have different "ranges." One camera might be able to see all objects in its immediate container, another might see only the objects in immediately adjacent positions. A two-way radio is a sensor that has access to only one other object: the other radio it is paired with. Range can be defined in terms of containment too: a visual sensor like a camera generally cannot see objects contained inside of other objects (e.g. objects contained in a box). An X-ray sensor can see through containers unless they are made of lead.

The features reported also depends on the type of the sensor. The Truckworld has a predefined set of *visual* features, for example, including color, type, and position. A camera sensor can be designed to report on those features only. Other sensors are more specific: sonars report position only, scales report bigness only. Most sensors report accurately on an object's position.

Sensors also have "noise" parameters causing them to distort a property's value; the nature and magnitude of the distortion depends on the feature and on the particular sensor. Distortion can also depend on the prevailing state of the world: visual sensors tend to work better when it's light outside; radios can distort their messages if used during electrical storms, etc.

### 3.1 Built-in sensors

The truck has several built-in sensors that provide information about the truck's internal state and the current location of the truck. These built-in sensors are:

- `<bay-name>-sensor`
  Senses the contents of the cargo bay.
- `<arm-name>-sensor`
  Senses the objects that the arm is currently holding.

- `speed-sensor, heading-sensor, status-sensor, odometer-sensor`
  Senses the value of the truck's *current* speed, heading, status, and odometer, respectively.
- `fuel-tank-sensor`
  Senses the amount of fuel in the fuel tank.
- `truck-sensor`
  Senses the objects at the truck's current location.

The truck manipulates these sensors directly, using the `truck-set` and `truck-read` commands.

## 3.2 Sensing examples

Sensing is a three-step process:

1. Setting the sensor causes it to start gathering information.
2. A period of time elapses while the object does the gathering. (We tend to set things up so sensors that provide better and more detailed information about the world take longer to gather it.)
3. The agent reads the sensor, providing it with the information just gathered.

For our first example, we have positioned the truck in the SENSOR-NODE location, and we are now ready to use the built-in `truck-sensor` to sense the other objects at that location.

```
truckworld> (truck-set truck-sensor)
truckworld> (wait 1)
truckworld> (truck-read truck-sensor)
COMMAND FROM SERVER: (SENSOR TRUCK-SENSOR
                             (((POSITION 0) (KIND ROADSIGN) (DIRECTION N)
                                  (CONDITION GOOD) (LENGTH 20))
                              ((POSITION 1) (KIND ROADSIGN) (DIRECTION NE)
                                  (CONDITION GOOD) (LENGTH 20))
                              ((POSITION 2) (KIND ROCK))
                              ((POSITION 3) (KIND ROCK))
                              ((POSITION 4) (KIND ROCK))
                              ((POSITION 5) (KIND ROCK))
                              ((POSITION 6) (KIND FUEL-DRUM))
                              ((POSITION 7) (KIND FUEL-DRUM)
                              ((POSITION 8) (KIND FUEL-DRUM)
                              ((POSITION 9) (KIND COLOR-SENSOR)
                                  (SENSOR-ID SENSOR-47))
                              ((POSITION 10) (KIND BAD-COLOR-SENSOR)
                                  (SENSOR-ID SENSOR-841))
                              ((POSITION 11) (KIND SONAR)
                                  (SENSOR-ID SENSOR-51))
                              ((POSITION 12) (KIND X-RAY-MACHINE)
                                  (SENSOR-ID SENSOR-845))
                              ((POSITION 13) (KIND LEAD-BOX))
                              ((POSITION 14) (KIND BOX))
                              ((POSITION 15) (KIND SOUNDPROOF-BOX))))
```

The default `truck-sensor` reports on some basic properties of objects in the same container (location) as the truck. One property is the position of each object within the location. Another is the object's *kind*, e.g. road sign, fuel drum. The sensor report delivered to the agent also contains information identifying the sensor that generated the information.

Of special note are the `road sign` objects located at positions 0 and 1. Every location contains a road sign object for every road that connects to it. "Written" on each sign is some information about the connecting road. The information includes the direction of the road (i.e. what the truck's heading has to be in order to travel on that road), the current condition the road, and the road's length. All of these features are available to the `truck-sensor`

Our second example demonstrates how to manipulate external sensors (i.e. sensors that are not built into the truck). External sensors can be manipulated by the truck's arms, using the `arm-set` and `arm-read` commands. Here we use `arm-1` to grasp the `color-sensor` located at position 9 of the current location; we then set it, then wait, then read its state:

```
truckworld> (arm-move arm-1 outside)
truckworld> (arm-move arm-1 9)
truckworld> (arm-grasp arm-1)
truckworld> (arm-set arm-1 0)
truckworld> (wait 5)
truckworld> (arm-read arm-1 0)
COMMAND FROM SERVER: (SENSOR SENSOR-47
                             (((POSITION 0))
                              ((POSITION 1))
                              ((POSITION 2) (COLOR RED))
                              ((POSITION 3) (COLOR RED))
                              ((POSITION 4) (COLOR GREEN))
                              ((POSITION 5) (COLOR CYAN))
                              ((POSITION 6))
                              ((POSITION 7))
                              ((POSITION 8))
                              ((POSITION 10))
                              ((POSITION 11))
                              ((POSITION 12))
                              ((POSITION 13))
```

```
                                 ((POSITION 14))
                                 ((POSITION 15))
                                 ((POSITION 16))))
```

The 0 parameter to the `arm-set` and `arm-read` commands refers to the fact that the sensor's new position is 0, within the arm bay associated with `arm-1`. The good color sensor takes 5 time units to gather information.

The color sensor reports only the position and color of objects. Note that the report does not mention the object's `kind`. Some objects (e.g. roadsigns) do not have a color property, in which case the sensor reports the object's position only. Luckily for us, rocks have color. The first 2 are red, and the second 2 are green. Sensing color again produced the following report:

```
truckworld> (arm-set arm-1 0)
truckworld> (wait 5)
truckworld> (arm-read arm-1 0)
COMMAND FROM SERVER: (SENSOR SENSOR-47
                                (((POSITION 0))
                                 ((POSITION 1))
                                 ((POSITION 2) (COLOR RED))
                                 ((POSITION 3) (COLOR RED))
                                 ((POSITION 4) (COLOR GREEN))
                                 ((POSITION 5) (COLOR LIGHT-GREEN))
                                 ((POSITION 6))
                                 ((POSITION 7))
                                 ((POSITION 8))
                                 ((POSITION 10))
                                 ((POSITION 11))
                                 ((POSITION 12))
                                 ((POSITION 13))
                                 ((POSITION 14))
                                 ((POSITION 15))
                                 ((POSITION 16))))
```

Notice that the color of the object at position 5 was originally `CYAN`, but now is `LIGHT-GREEN`. Has the rock changed color? Although it's possible that another agent or an exogenous event changed the object's color between the two reports, in this case the discrepancy is due to sensor noise. This particular color sensor has been designed to distort the actual color of the objects a little bit, 20 percent of the time. (The color feature is defined as an ordered collection of symbolic values, and 20% of the time the sensor reports a color adjacent in the collection to the true color.)

For our third example, let's try the `bad-color-sensor` at position 10, which has much worse noise characteristics, but also takes only 1 time unit to process information. This example also demonstrates that we can issue the `arm-set` and `arm-read` commands without grasping the object (Refer to Section 5); instead we just move an arm to the same position as the sensor.

```
truckworld> (arm-move arm-1 10) ;;; where the sensor resides.
truckworld> (arm-set arm-1 nil)
truckworld> (wait 1)
truckworld> (arm-read arm-1 nil)
COMMAND FROM SERVER: (SENSOR SENSOR-841
                             (((POSITION 0))
                               ((POSITION 1))
                               ((POSITION 2) (COLOR PINK))
                               ((POSITION 3) (COLOR ORANGE))
                               ((POSITION 4) (COLOR BLUE))
                               ((POSITION 5) (COLOR PURPLE))
                               ((POSITION 6))
                               ((POSITION 7))
                               ((POSITION 8))
                               ((POSITION 11))
                               ((POSITION 12))
                               ((POSITION 13))
                               ((POSITION 14))
                               ((POSITION 15))
                               ((POSITION 16))))
```

The next example involves the sonar at position 11, and demonstrates how *containment* can affect what objects a sensor reports on. The sonar sensor reports only on whether there is an object at a position, but it is able to penetrate containers at the location, e.g. the boxes at position 13 and 14.

```
truckworld> (arm-move arm-1 11)
truckworld> (arm-set arm-1 nil)
truckworld> (wait 1)
truckworld> (arm-read arm-1 nil)
COMMAND FROM SERVER: (SENSOR SENSOR-51
                             (((POSITION 0)) ((POSITION 1)) ((POSITION 2))
                              ((POSITION 3)) ((POSITION 4)) ((POSITION 5))
                              ((POSITION 6)) ((POSITION 7)) ((POSITION 8))
                              ((POSITION 9)) ((POSITION 10)) ((POSITION 12))
                              ((POSITION 13)) ((POSITION 13 0))
                              ((POSITION 14)) ((POSITION 14 0))
                              ((POSITION 15)) ((POSITION 16))))
```

Notice the reports associated with positions 13 and 14: they indicate both that there is an object at those locations and there is also an object contained *within* each box, both at position 0. The box at position 15 is a sound-proof box, so the sonar can't penetrate it.

SENSOR-NODE also contains an `x-ray-machine`, which can "see" through all types of containers except for `lead` containers. We invite you to play with this sensor.

### 3.3 Sensor types

The sensors described so far gather information when they are `set`, and report that information when they are `read`.

A sensor that gathers when `set` is called a *passive* sensor. The alternative is an *active* sensor, which gathers information when some aspect of its environment changes. A tape recorder is an active sensor that gathers information whenever an object in its container generates a sound. A motion detector is an active sensor that gathers information whenever an object is added to or deleted from its container.

Sensors also differ on what they do with the information once they gather it. The sensors described in the examples are all *recording* sensors: they report on sensed information when they are `read`. Alternatively, *broadcasting* sensors send information over a channel as soon as they gather it. Motion detectors are broadcasting sensors because they generate a loud sound when they sense a change to their container. Two-way radios are passive broadcasting sensors: they gather information using a `set` operation, but immediately broadcast it to the other radios on the same frequency.

Two trucks at the same location can "speak" to each other using loudspeakers and microphones. A loudspeaker is a passive, broadcasting sensor that is `set` with a string argument, and generates a "sound" consisting of that string. A microphone is an active broadcasting sensor that receives the string and immediately communicates it to its owner truck. Alternatively a truck could plant a tape recorder (an active recording sensor) at the location it expects a message, and drop by later to `read` that message.

## 4 Running a simulation

Truckworld is intended to be used in a client/server sort of relationship: a single server process runs the simulation itself, then various truck clients can connect to the simulation, communicating with the simulation using TCP/IP sockets. The clients and server all run asynchronously.

[2] provides details on how to run the client and server in separate process spaces, and how to run a simulation with multiple agents. For demonstration purposes we provide a single-process version of the Truckworld: a single client runs synchronously with the simulator. We also provide a simple demonstration world with some interesting objects.

There are really three parts to a Truckworld simulation:

- The simulation itself: it takes commands in from its trucks, changes the world state, and sends information back to the clients about changes in the world. There is one simulator, regardless of the number of clients.
- For each client:
  - A command processor: a function that takes commands from some source and communicates them to the simulator. The command processor also gets information back from

the simulator (e.g. reports from the sensors) and returns them to the client. Our demonstration command processor prompts and accepts commands from the human user, and prints any return messages from the simulator.

  Alternative command processors read or write commands form a file, or provide an interface between the simulator and a planning program that generates the commands automatically.

  – A displayer: Figure 1 shows the graphic displayer. A displayer is a depiction of the simulated world's state, generally providing more detail about the client it is attached to. Every time the simulator changes the world it sends a message to every active displayer describing the change. Each graphic displayer changes its display window accordingly. Alternative displayers write these messages to a file or print them or ignore them altogether. Note that the displayer is supposed to be an aid to the *human*; the main communication between the agent and the simulation happens in the command-processor loop.

Most of the simulator uses standard portable Common Lisp. Cooler command processors and displayers require some additional software:

- Running the clients and server in separate process space requires a TCP/IP package that makes calls to implementation-dependent code. Our current version runs under Allegro CommonLisp Version 4.1.
- The graphic displayer requires CLX
- The Truckdriver command processor (not described in this document) requires CLIM.

## 4.1 Running a demonstration world

The standard demo package runs in a single process space, but uses the graphic displayer, thus requires CLX. Here's how to load it:

1. Get the Truckworld code (see below).
2. Locate the file `loader.lisp` in that directory. You'll need to change variables at the top of this file to provide an absolute pathname for the directory containing the simulator.
3. Load the file `loader.lisp`.
4. Execute the form `(load-integrated-simulator)` which will load all necessary files.
5. Execute the form `(run-demo :display <display>)` to start the simulator. The argument `<display>` is the hostname of the machine on which you want the displayer's window to be displayed, as a string. This is an optional argument, and should be of the form `machinename`, not `machinename:0` or anything like that. If it is not supplied, no display is created.

At that point the display window will come up and a command-processor loop will start. You can type any Truckworld commands, or `(bye)` to terminate.

## 4.2 Cheating

You can easily get your truck in trouble, generally either by running out of gas or rolling the truck off a cliff, or getting stuck in the mud. You'll know you're out of gas when the simulator won't let you travel and the fuel gauge looks empty. Use the `fuel-tank-sensor` to be sure. You'll know you're seriously damaged when a status indicator different from `HAPPY` gets displayed.

Though you should really just be more careful, here's how to cheat your way out of trouble:

1. Break your way out of the command processor. A control-C or two usually does the job.
2. At that point you're at the Lisp prompt, and you can execute the functions
   - (cheat-fill-fuel-tank) — fill up the fuel tank as if by magic!
   - (cheat-reset-status) — make the truck's status `HAPPY` again

At that point you can continue the command loop (`:CONTINUE` should work) and start executing Truckworld commands again.

## 4.3 Getting the Truckworld code

Send mail to `truckworld-users-request@cs.washington.edu` to get the code. Two mailing lists facilitate communication among Truckworld users:

- `truckworld-users@cs.washington.edu` is used to announce new releases, bug fixes, and so on. Anybody requesting Truckworld code is put on this list. Messages to this list should be of interest even to casual users.
- `truckworld-debuggers@cs.washington.edu` is for hard-core Truckworld hackers. Send bug reports or detailed questions to this address.

# 5 Command Summary

Here's a short reference guide to all the Truckworld commands. Note that `(bye)`, `(wait <n>)` and the `cheat` commands aren't part of the truck's repertoire of commands, but specify interactions between the simulator and the demonstration command interface.

## 5.1 Arm commands

- `(arm-move <arm-name> folded)`
  Can be executed any time.

- `(arm-move <arm-name> outside)`
  If the arm is folded, the arm moves just outside the truck. Otherwise, it moves the arm to the outside of its current container.

- `(arm-move <arm-name> inside &OPTIONAL <position>)`
  If the arm is folded, moves the arm inside the truck, readying it to move to one of the truck's "special positions". If the arm is positioned just outside a container, it moves inside the container. If `<position>` is provided, it moves to that position within the container, otherwise moves to an arbitrary position.

- `(arm-move <arm-name> <position-index>)`
  Arm must be inside some container object (e.g. outside of the truck), and not folded. Move arm to that position within its current container.

- `(arm-move <arm-name> <bay-name> &OPTIONAL <position>)`
  Arm must be inside the truck. Bay name is one of the truck's cargo bays, or `tire-bay` or `weapon-bay`. If `<position>` is supplied, moves to that position within the bay, otherwise moves to an unspecified position within the bay.

- `(arm-move <arm-name> fuel-tank)`
  Arm must be inside the truck. Move to the fuel tank.

- `(arm-grasp <arm-name>)`
  Arm must be at a position within a container, which must contain an object. Arm must have the capacity to hold the object. Transfer the object from its current container to the corresponding arm bay. This operation may fail according to the arm's *clumsiness* parameter.

- `(arm-ungrasp <arm-name> <position>)`
  Arm must be at a position within a container, which must not contain an object. The named position of the arm bay must contain an object.

- `(arm-pour <arm-name> <position>)`
  Transfer fluid from the vessel at the named position in the arm's bay to the vessel at the arm's current position. Transfer until the source is empty. If the destination won't hold all the fluid, the excess is lost.

- `(arm-set <arm-name> <position-or-NIL> &REST <args>)`
  Set a property of the object at the arm's current position (NIL) or the object at that position in the arm's bay. Effects depend on the object being set.

- `(arm-read <arm-name> <position-or-NIL>)`
  Read the information stored in the sensor object at the arm's current position (NIL) or the sensor at that position in the arm's bay. Returns nothing if the object is not a sensor.

## 5.2 Truck commands

- `(change-speed <speed>)`
  Change the value of the speed controller to `stop`, `slow`, `medium`, or `fast`. Note this does not

cause movement, but dictates how fast the truck will move when a `truck-move` command is issued.

- `(change-heading <heading>)`
  Change the value of the speed controller to one of `N`, `NE`, `E`, `SE`, `S`, `SW`, or `W`. Again, this does not cause motion, but influences the outcome of `(truck-move)`.

- `(truck-move)`
  Arms must be folded, heading must be in a direction of a road, speed must not be `stop`, must have sufficient fuel. Move down the road, where the effects and the amount of time consumed depend on characteristics of the truck and of the road.

- `(truck-set <sensor-name>)`
  Activate one of the truck's internal sensors: `fuel-tank-sensor`, `weapon-bay-sensor`, `tire-bay-sensor`, `heading-sensor`, `speed-sensor`, `<bay-name>-sensor`, `<arm-name>-sensor`, `truck-sensor`.

- `(truck-read <sensor-name>)`
  Read the sensor after activating it. An intervening `wait` is usually necessary.

### 5.3 Commands used by the command interface

- `(bye)`
  Ends the demonstration.

- `(wait <n>)`
  Wait `<n>` time units. Generally wait for a process (like sensing) to finish.

## References

[1] Dat Nguyen, Steve Hanks, and Chris Thomas. The Truckworld Manual. Technical report, University of Washington, Department of Computer Science and Engineering, 1993. Forthcoming.

[2] Chris Thomas, Steve Hanks, and Dat Nguyen. Multiprocess and Multiagent Operation of the Truckworld, 1993. Forthcoming; supplied with Truckworld code.