ICFP 2008 Programming Contest
Task Description

(Version 1.5)

1 Overview

Recent breakthroughs in higher-order, statically-typed, metaspatial communication will enable data to be transferred between Mars and Earth almost instantaneously. As such, NASA is seeking examples of real-time control software to operate its latest model Martian rovers from control centers on Earth. Since it is well known that the ICFP contest attracts the *crème de la crème* of programmers from around the world, NASA has decided to use the current contest as a means of gathering software prototypes for their new control system. We are pleased to announce that this year's winning entry will in fact be used to control the rover on NASA's very next mission to Mars![1]

Your control software will communicate with the rover over a network socket. Its object is to guide the rover safely from a given starting location to its home base. The controller's primary function is to avoid the boulders and craters that litter the Martian surface. As an added nuisance, the local inhabitants, who are decidedly hostile, will immediately destroy any rover they can get their fourteen sticky fingers on. Note that Martians, like dogs, vary in intelligence.

Control software prototypes will be evaluated according to their performance in a series of trials, the details of which are given below. Each trial consists of five runs on a given region of Mars. As a means of preparing for these trials, this document and its accompanying software provide sufficient details and infrastructure for the development of prototype candidates. Good luck, and may yours be the winning entry, to be used on Mars itself.

2 Mars rover behavior

The rover is a circular vehicle of radius 0.5m, which your controller must guide across the Martian terrain. To do so, your controller must establish a connection to the rover over a TCP/IP socket.

[1]Subject to budget constraints.

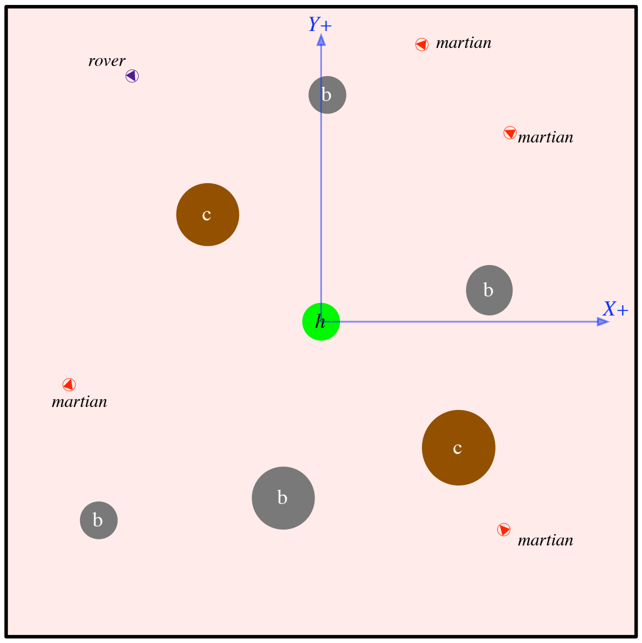

Figure 1: A region of Mars

This socket it used for all communication to and from the rover. While metaspatial communication is very fast, there is some latency in the connection (on the order of 75 microseconds).

Once the connection is established, the controller will be sent an initial message about the dimensions of the world and the physical characteristics of the vehicle. The world is modeled as a rectangular segment of the xy-plane centered at the origin. The *home base* — the rover's intended destination — is a circle (radius 5m) located at the origin. The vehicle's characteristics include its maximum speed, its maximum rotational speed when turning, and a few other facts. NASA is testing various different models of rovers, which have varying performance. They are also testing different regions of Mars, with a wide range of characteristics. For a given trial, the rover's performance and the map will be fixed, but these will vary from trial to trial. Complete information on the initial message is furnished in Section 3.1.1 below.

About one second after the initial message is sent, the first run starts and the server begins sending a stream of vehicle telemetry data to the controller. Telemetry data consists of location, speed, and information about the local terrain (see Section 3.1.2 for full details). At any time after the telemetry data has started streaming to the controller, the controller may issue commands back to the server to direct the vehicle towards home base.

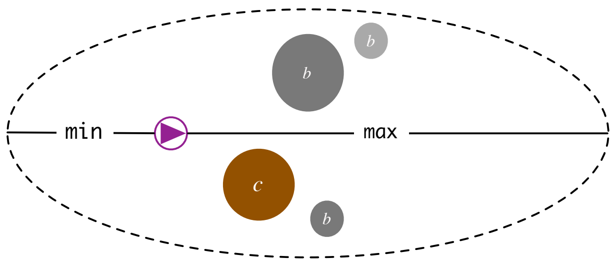

Figure 2: A sketch of the vision model

The rover must avoid three kinds of perils on its way home. If the rover hits a boulder, or the edge of the map, it bounces off and loses speed. Hitting a boulder happens if the distance between the center of the boulder and the center of the rover is less than the sum of their radii. If the center of the rover enters a crater, it falls in and explodes. If a vehicle is caught by a Martian, it is destroyed and its remains are sacrificed to the Gods of Barsoom, of whom it is best not to speak further.

Martians, while hostile, possess no special physical abilities; that is, they cannot drive through craters, pass through objects, or escape the boundary of the map. The physics of Martian movement is the same as for the rover, although they may be faster or slower, *etc*. Martians are slightly smaller than the rover, having a radius of 0.4m.

An illustration of a typical Martian region appears in Figure 1. Boulders, craters and home base are marked *b*, *c* and *h* respectively.

2.1 Vision

The rover's visual sensors cover an elliptical area that extends further in the direction that the rover is facing. Figure 2 depicts this region. The rover is oriented toward the right in this illustration. Implicitly, there is an ellipse defined by min and max, with the rover always positioned at one of the foci. The rover can see everything within this ellipse, with the exception of those objects that are occluded by boulders. In the figure, the rightmost boulder is not visible to the rover because of the larger boulder blocking it. The lowermost boulder, on the other hand, is visible, since craters do not occlude vision.

2.2 Speed

The linear speed of the rover at time t' ($s_{t'}$) is computed according to its speed at its predecessor time t according to the following formula.

$$s_{t'} = \max(s_t + (t' - t)a - k(t' - t)s_t^2,\ 0)$$

The latter term is the simulated *drag* on the vehicle. Note a, the *acceleration*, can be negative if the rover is braking. The rover's acceleration and braking rates are not known, although they can be determined by experiment. The effect of drag is to limit the maximum speed of the rover. The maximum speed is known (it is communicated as part of the initial message), but the drag coefficient is not known.

2.3 Turning

The rover has two turning speeds — regular turns and hard turns — in both directions. When the rover receives a commend to turn left, its turning state moves one "notch" in the leftward direction; likewise for right. Note that while the turn rate and hard-turn rates are known, the rotational acceleration of the vehicle is finite and thus it will take some time change from one turning mode to another. Section 3.2 addresses the mechanics of steering messages in greater detail.

3 Network protocol

Communication between the server and controller will be over a TCP/IP socket using plain-text messages encoded in ASCII. The controller will be given a server hostname and port number as command-line arguments at the beginning of each trial. The controller should establish a client-side TCP/IP connection to the server; this socket will be used by the controller to send commands to the vehicle and by the server to send telemetry data to the controller.

A *message* is a sequence of tokens delimited by the ASCII space character (`0x20`) and terminated by a semicolon. The tokens in a message represent quantities of various kinds and have the following formats:

- Distances, lengths, and locations (x and y coordinates) are given in meters in fixed-point representation rounded to the nearest thousandth (millimeter).
- Angles are given in degrees in fixed-point representation rounded to the nearest tenth.
- Angular velocities are given in degrees per second in fixed-point representation rounded to the nearest tenth.

- Speeds are given in meters per second in fixed-point representation rounded to the nearest thousandth.
- Durations are given in whole milliseconds (since the start of the simulation).

3.1 Messages from the server to the controller

There are a variety of messages that the rover sends to the controller. Each message begins with a single character that denotes the message kind.

3.1.1 Initialization

The exact characteristics of the vehicle are unspecified and may differ between trials, but information about the vehicle will be given at the beginning of each trail. Once the connection to the server is established, the controller will receive an initial message with the following format:

`I` *dx dy time-limit min-sensor max-sensor max-speed max-turn max-hard-turn* `;`

`I` is the message tag signifying initialization.

dx is the span of the map's x-axis (meters). A map with *dx* 100.000 extends from -50.000 to 50.000 on the x-axis.

dy is the span of the map's y-axis (meters). A map with *dy* 100.000 extends from -50.000 to 50.000 on the y-axis.

time-limit is the time limit for the map (milliseconds). Map time limits are discussed in Section 4 below.

min-sensor is the minimum range of the vehicle's visual sensors (meters). See the discussion of the vision model in Section 2.1 above.

max-sensor is the maximum range of the vehicle's visual sensors (meters). See the discussion of the vision model in Section 2.1 above.

max-speed is the maximum speed of the vehicle (meters per second).

max-turn is the maximum rotational speed when turning (degrees per second).

max-hard-turn is the maximum rotational speed when turning hard (degrees per second).

3.1.2 Telemetry stream

During a run, the server sends a steady stream of telemetry data to the vehicle controller (roughly one message every 100 milliseconds). This data includes information about the vehicle's current state (control-state, heading, velocity, *etc.*) as well as information about the local map conditions (obstacles and enemies).

`T` *time-stamp vehicle-ctl vehicle-x vehicle-y vehicle-dir vehicle-speed objects* `;`

where

`T` is the message tag signifying telemetry data.

time-stamp is the number of milliseconds since the start of the run.

vehicle-ctl is the current state of the vehicle controls. It is a two-character sequence with the first character specifying the acceleration state (`a` for accelerating, `b` for braking, or `-` for rolling, *i.e.*, moving at a constant speed) and the second character specifying the turning state (`L` for hard-left turn, `l` for left turn, `-` for straight ahead, `r` for right turn, and `R` for hard-right turn). Note that the rover will gradually slow down when rolling, because of drag.

vehicle-x is the x-coordinate of the vehicle's current position.

vehicle-y is the y-coordinate of the vehicle's current position.

vehicle-dir is the direction of the vehicle measured as a counterclockwise angle from the x-axis.

vehicle-speed is the vehicle's current speed (meters per second).

objects is a sequence of zero or more obstacles and/or enemies that are visible to the vehicle. An item is *visible* if it falls in the range of the vehicle's visual sensors; recall that range is part of the rover characteristics given in the server's initial message. Object messages have two different formats depending on the type of object. If the object is a boulder, crater, or home base, the format is

object-kind object-x object-y object-r

where

object-kind is one of `b` (for a boulder), `c` (for a crater), or `h` (for home base).

object-x is the x-coordinate of the object's center.

object-y is the y-coordinate of the object's center.

object-r is the radius of the object.

If the object is a Martian, the description has the format

`m` *enemy-x enemy-y enemy-dir enemy-speed*

Here is an example telemetry message. Note that we have split this message over multiple lines to improve readability — the actual message would not contain any newline characters.

```
T 3450 aL -234.040 811.100 47.5 8.450
  b -220.000 750.000 12.000
  m -240.000 812.000 90.0 9.100 ;
```

This message describes the vehicle's state at 3.45 seconds after the start of the run. It is currently accelerating and turning hard to the left. Its position is $(-234.040, 811.100)$, its direction is 47.5 degrees (roughly NE), its velocity is 8.450 meters per second, and it sees one boulder and one Martian.

3.1.3 Adverse events

There are also messages to signal unhappy occurrences. These messages have the format:

message-tag time-stamp **`;`**

where the *message-tag* is one of

`B` for a crash against a boulder or the map edge. When the rover crashes into a boulder, it bounces off and loses speed. Each crash message is immediately followed by a telemetry message so that the controller can update its state.

`C` if the vehicle fell into a crater. Falling into a crater destroys the rover and ends the run.

`K` if the vehicle was killed by a Martian, which ends the run.

3.1.4 Success message

The server sends the message "**`S`** *t* **`;`**" when the vehicle reaches home base safely. The current run is terminated on success.

3.1.5 End-of-run message

At the end of a run, the server sends the message "**`E`** *t s* **`;`**," where *t* is the time since the beginning of the run, and *s* is the score (*i.e.*, the run time plus any penalties). Note that each run will end with exactly one of the following sequences of server messages:

- **`C`** *t* **`;`**, then **`E`** *t s* **`;`**
- **`K`** *t* **`;`**, then **`E`** *t s* **`;`**
- **`S`** *t* **`;`**, then **`E`** *t s* **`;`**
- **`E`** *t s* **`;`** preceded by none of **`C`**, **`K`**, or **`S`**, indicating that the time limit has been reached

Once a run has terminated, there will be a pause of at least one second before the start of the next run. Note that the controller should not exit at the end of the run, but instead should prepare for another run. Also note that the initialization message described in Section 3.1.1 is only sent once per trial. Each run of the trial uses the same map, although the rover's initial position and the number and location of Martians can vary from run to run.

3.2 Messages from the controller to the server

The rover behavior is controlled by a pair of state machines (Figure 3), which, in turn, are controlled by commands sent by the controller. Each command consists of an optional acceleration (**`a`**) or braking (**`b`**) command, followed by an optional turning command (**`l`** for left and **`r`** for right), and followed by a semicolon (**`;`**). Thus, the grammar of controller-to-server messages is

$$\mathit{Message} ::= \texttt{;} \mid \texttt{a;} \mid \texttt{b;} \mid \texttt{l;} \mid \texttt{r;} \mid \texttt{al;} \mid \texttt{ar;} \mid \texttt{bl;} \mid \texttt{br;}$$

No other characters (including whitespace) should be sent over the command stream!

While communication with the rover is fast, there may be some latency (less than 20 milliseconds) in processing the commands. The controller may send messages as often as it likes, although flooding the network can negatively affect performance. We recommend only sending messages in response to telemetry data, although you may need a sequence of messages to reach the desired control state.

4 Contest organization and scoring

The contest is run as a series of trials of varying difficulty. A *trial* consists of five *run*s on the same map. Each map has an associated *time limit* of some number of milliseconds. A *limit-n map* has an upper limit of n milliseconds.

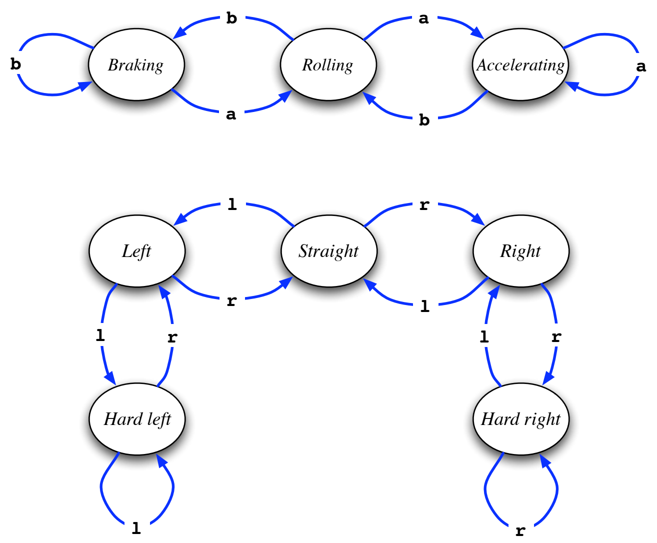

Figure 3: Vehicle-control state machines

The *score* for a run is the amount of time it takes to complete the run or be destroyed, plus any penalties.

- If a rover reaches home base on a given limit-n map in some $t \leq n$ number of milliseconds, the run score is t.
- If the rover fails to reach home base on a given limit-n map, the run score is given by the equation $2n - t + p$, where t is the elapsed time, and p is the penalty as follows:
  - 100 if the time limit has been exceeded,
  - 600 if the rover was destroyed by a Martian, or
  - 1000 if the rover fell into a crater.

  Note that t is at most n in this formula, since the run is halted when t exceeds n. Therefore $(2n - t)$ will always be between n and $2n$ inclusive.

As in ski racing, lower scores are better.

The *trial score* is the sum of the three lowest scores in the trial.

The winner of the contest will be determined by a series of *heats*. A heat consists of all remaining competitors being subjected to the same trial. After each heat, the competitors are ranked by their trial score and some fraction of the better competitors will advance to the next heat, while the remaining competitors will be eliminated. Heats will continue until there is a single winner remaining. From one heat to the next, the given trial may differ arbitrarily.

Programs that core dump will be disqualified.

Programs that attempt to subvert the host or server, or that generate illegal messages will be disqualified.

5 Submission instructions

Your submission must run in the Linux environment provided by the LiveCD provided on the contest web site. The LiveCD includes many popular language implementations, but if your favorite language is not included, you may still submit a solution. The only restriction is that it must run in the LiveCD environment. Details about the LiveCD can be found at

`http://icfpcontest.org/live-cd.html`

Contest entries must consist of a single gzipped tarball named `icfp08.tgz`. You submit your entry using the web form at

`http://icfpcontest.org/submit.php`

The first time that you submit your entry, you will be given a unique identifier that you must use for any resubmissions.

After unbundling, an optional installation script is run. The resulting directory tree must include all of the following:

- at the top level, a directory `icfp08`.
- a file `team`, inside `icfp08`, which consists of a single line of text, the team name. Your team name must be no longer that 63 ASCII characters.
- a file `contact`, inside `icfp08`, which consists of the team member names and email addresses in the following format:

  ```
  John Doe <johndoe@gmail.com>
  ```

  Name/address pairs must appear one per line.

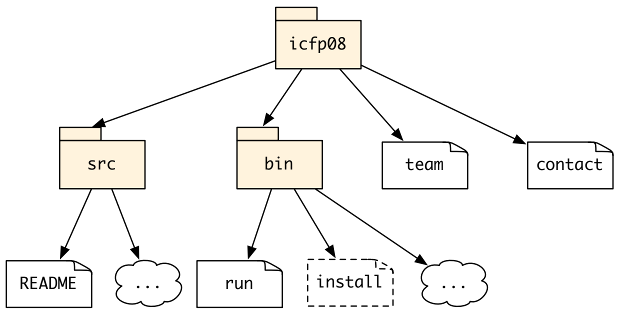

Figure 4: Submission directory structure

- a directory `src` containing all the source code for your contest entry. Inside `src`, your code may be organized however you like. Please note: you *must* provide source code, even if you use an unsupported language or compiler to develop your entry. Submissions with no accompanying source code will be disqualified.
- a file `README`, inside `src`, to assist the judges' understanding of your code. If you used tools not among those provided on the LiveCD, specify the language and compiler your team used in this file and describe how to obtain and install the tools.
- a directory `bin`, inside `icfp08`, containing whatever executables and scripts support the running of your controller.
- an executable file `run`, inside `bin`, which runs your client. It may be an executable, or it may be a shell script that manages running your client. It must take two command-line arguments, the server's hostname and the port number, in that order. That is, the judges must be able to execute your client with

```
bin/run hostname port
```

  from inside the `icfp08` directory.

Your tarball may include other files as well, including code for general-purpose third-party libraries, with your submission as long as your `README` enumerates those libraries. Teams may not use code written by other teams working on the contest as a "library." Only generic libraries, *e.g.*, one that provides matrix operations, may be used. Teams who use libraries as a means of sharing code with one another will be disqualified.

The structure of a contest entry is illustrated in Figure 4.

In addition, your archive may contain an optional installation script `install` in the `bin` directory. If this script is present, it will be executed using the command

```
bin/install
```

from inside the `icfp08` directory. The `PWD` shell variable and `pwd` command can be used to determine the path to the `icfp08` directory (and thus to your scripts, *etc*.). The layout of your submission is not checked until after this program is run, so it may be used to generate or modify the files in your submission, but it should not attempt to copy files outside the `icfp08` directory.

The process of running your client for a given trial will involve the following steps:

```
tar -xzf icfp08.tgz
cd icfp08
if test -r bin/install ; then
  chmod +x bin/install
  bin/install
fi
chmod +x bin/run
bin/run hostname port
```

These commands will be run as user `knoppix` (*not* `root`) in a temporary directory.

The specifications above must be matched exactly, since contest entries will be processed by scripts depending on the exact structure presented here. Failure to meet the specifications is grounds for disqualification.

6 How to test your program

NASA is providing a Martian simulator and sample maps for contestants to test their code on while developing their contest entries. Note that while this simulator is a physically accurate simulation of the Martian environment, the vehicle characteristics may vary in the actual trials. Furthermore, the environment used to test your controller may be harsher (*i.e.*, more obstacles) than the samples and the Martians therein may be faster and smarter. Prepare for the worst!

The sample simulator and maps are available for download from

```
http://smlnj.org/icfp08-contest/simulator.html
```

To run the server, you must supply it with the name of a map file. Details on the map file format appear on the web site.

7 Implementation hints

Because the controller program is sensitive to network latency, you should disable Nagle's algorithm for the socket. You can do this using the `setsockopt` system call with the `TCP_NODELAY` option.

Implementations may use information gathered in early runs of a trial to improve their score in later runs. Note that since a single execution of the controller program is used for the whole trial, you do not need to use disk storage to communicate information between runs. No information can be communicated between trials.

Good luck!

Document history

Version 1.0 Initial version.

Version 1.1 Changed installation behavior to make the `install` and `run` scripts executable before running it.

Version 1.2 Correction in Section 3: “IP address” changed to “hostname.”

Version 1.3 Fixed small text typo. Clarified the fact that drag is unknown. Fixed installation behavior description. Clarified falling and crashing behavior.

Version 1.4 Removed erroneous definition of a run from Section 4. Stated that generic third-party libraries *may* be submitted in Section 5.

Version 1.5 Stated that angular velocity (degrees per second, as in *max-turn* and *max-hard-turn*) are given to the nearest tenth of a degree.