Mars Domain Spec v0.1
Matt Schmill
12/2/2008

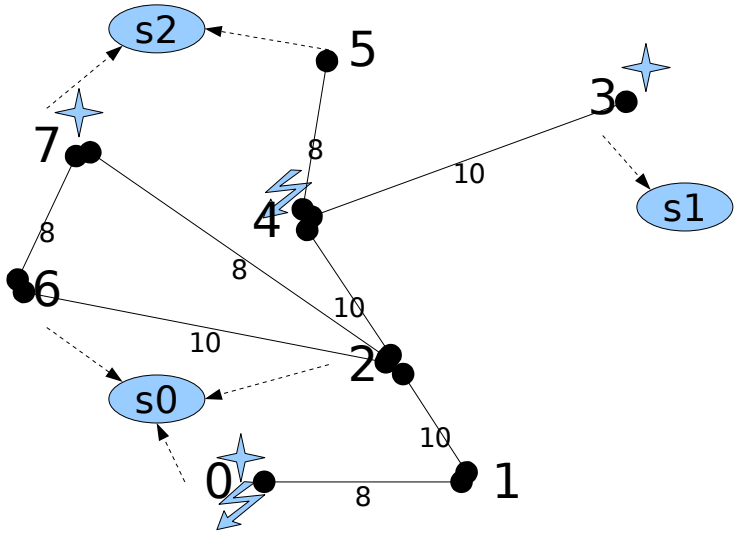
Mars Domain Description

The mars domain is an abstract, discrete, simulated environment into which agents can be placed. It is built using the MonCon Substrate. The domain consists of the following structures:

- a "map", which is an undirected graph of "waypoints"
- a set of "science points", representing geographic features on Mars that can be studied by agents
- boolean vectors that define what actions are legal for agents to take at the various waypoints (such as recharging or localizing)
- a table of agent properties, for agent properties that are determined by the environment (and not the agent itself, such as "location")
- a set of agents that are operating in the domain
- a timeline, onto which agent and domain events can be scheduled

The default Mars configuration is as follows:

- 8 waypoints
- 3 science points
- 2 points where agents can recharge
- 3 points where agents can localize



Drawing 1: default mars domain map

Agents

The domain is populated by *agents*, and agents are controlled by *controllers*. The primary agent in the Mars simulation is the rover, and the primary controller used by the rover is called the ugpa controller. I cannot for the life of me remember what ugpa stands for.

Each MonCon agent (or at least the *good* ones) possess *observables* and *actions*. Observables are obviously anything the agent can observe and actions are things the agent can do to interact with the domain. Observables available to the rover are:

- power – battery power required to perform actions.
- bank – storage space for scientific actions.
- location – the waypoint at which the rover resides.
- cal – orientation (calibration) of the science platform toward a domain feature (science point).
- t_pano – time since the last panoramic image was taken.
- t_xmit – time since the last transmission to a relay station was made.
- error – the amount of sensor error due to operating with a simulated dead reckoning controller.
- status – privileged information; contains error codes returned by action controllers.

Actions the rover can take are:

- charge – restores the rover to maximum power when taken at a legal recharge waypoint.
- moveto(d) – changes the simulated location of the rover to *d* if it is possible to reach from the current location and if the rover has the necessary power.
- pano – takes a set of panoramic images of the current waypoint.
- science – performs scientific analysis of the currently calibrated science point.
- cal(s) – calibrates to a science point if it is possible to calibrate towards *s* from the rover's current location.
- loc – localizes, driving error to zero, when taken at a legal localize point.
- xmit – transmits the contents of the image bank to the nearest relay station. resets the image bank observable to the maximum quantity available and the rest is the relay station's problem.

Observables can be created to be noisy, breakable, or both. Currently, a parameterized uniform noise distribution is implemented, and a break-to-zero (the observable reports zero when it is broken) is implemented. Sensor breaks can be set to occur randomly, or they can be scheduled on the timeline.

The UGPA Controller

The primary rover controller as of this writing is called the ugpa controller. Look, I said I can't remember why. Maybe it stands for Unified Goal/Plan Architecture. That sounds right. Anyway, here's what an ugpa controller has:

- an action queue, which is a list of urgent-priority action specifications to be taken
- a goal queue, which contains goals the agent has been asked to achieve
- goals contain a target action and (possibly) a plan to allow the agent to execute the target action
- plans are themselves action queues, containing an ordered list of action specifications that will allow a goal to be achieved.

Now, the ugpa controller does this:

1. check the main action queue. if there is something there, do it.
2. else, check if there is a currently executing goal/plan. if there is, do the next thing in the plan.

3. else, check if there is a goal in the goal queue. if there is, pop off the front, build a plan, and set the currently executing goal.
4. else, don't do anything.

And that's essentially it. The only rub is that ugpa is MCL-enabled. So, after each action is executed, it calls out to mcl::monitor(). Any MCL recommendations are handled in control_react() (this is not unique to ugpa) and the ugpa controller puts repair actions on the main action queue, where they take priority over plans. When an existing goal is interrupted, the controller may specify what is to be done about the current plan: nothing (retry), replan, or abort.

The Timeline

The timeline allows us to schedule events on a deterministic schedule so that we can reproduce experimental conditions reliably to test the effects of changing system settings. The following can be scheduled on the timeline:

- domain events – changes to the map such as moving recharge/loc points, changing science point visibility, or changing the underlying reachability/move cost matrix.
- agent events – currently limited to sensor breakages.
- agent goals – goals may be issued to agents with controllers that can handle them.

Anomalies

Anomalies can currently be intentionally introduced into the Mars simulation in a limited number of ways.

- edge costs on the reachability matrix can be changed
  - edges can be deleted or moved entirely
- recharge points can be moved
- localize points can be moved
- the calibration/science matrix can be changed (changing the visibility of the science points)
- the costs of individual actions besides moveto can be changed

The net effect of most of this is that the underlying models used by agent controllers will be wrong. Due to this, the ugpa controller may make plans that fail to achieve the goal effect, but in any case any expectations generated on effected observables will cause a violation.

MCL Interoperability

The Rover is MCL-enabled. It uses its operator models (which are represented explicitly) to generate expectations about its action effects. Currently:

- moveto: “power” goes to MAX_NRG
- loc: “error” goes to 0
- xmit: “bank” goes to MAX_BANK && “t_xmit” goes to 0
- cal: “cal” changes
  note – this expectation is not specific because dynamically generated expectation parameters is

not implemented yet.

- pano: “bank” goes down by PANO_IMG && “t_pano” goes to 0
- science: “bank” goes down by SCI_IMG
- moveto: “loc” changes && “power” changes
  see note on cal

The Rover responds to MCL recommendations using actions maintained separately from the rover actions (they are subclasses of basic_repair_action). When a recommendation is made via the mcl::monitor() call, the Rover controller compares the recommendation code against its repair actions to determine which may be appropriate, then puts the correct repair on the main controller action q, preempting any plans that might currently be in progress. The Rover currently has the following repairs:

- rebuild
- sensReset
- effReset
- rescue

Declarations & Scoring

Moncon has a declaration system built in so that things can be declared and counted as they happen. Each of the repair actions are counted as they are taken, expectation violations are counted as they occur, etc. The declaration system is versatile and can be used as a basis for observables as well as to score instantiations of the mars rover. Logging of declarations is forthcoming.