Mission to Mars

Carl Ritson and Jon Simpson

Introduction

You are going to write the control program for a robot deployed to Mars. An operator will send commands to the robot over the interplanetary internet, where your program will be responsible for executing them and returning any results. It can take anywhere from minutes to hours for communication between the operator and robot (depending on the positions of Earth and Mars at the time) and as such, the robot must be semi-autonomous. For example, if an obstacle is detected in the path of the robot while it is in motion, it must stop before colliding, rather than waiting for operator input. The only alternative to this would be to make the the robot travel so slowly that the operator can always respond in time, but this would make progress agonisingly slow. The less operator input required (i.e. the more functions your program can provide autonomously) the faster the operator can work to complete a task [1].

The robot senses the world using a set of cameras: one for finding objects, and several others for detecting hazards. Special purpose sub-processors handle these camera feeds and provide information to your program - you don't have to write any image processing code for this assignment.

- The object camera provides a stream of object information, containing colour and bearing for everything in its field of view (90 degrees). Because object detection is processor-intensive, an object scan takes a bit of time (2 seconds).
- The hazard cameras produce risk assessments from various angles around the robot, they respond quickly but only provide a relative distance to the nearest detected hazard.

Information from the cameras will never be 100% accurate, especially whilst the robot is in motion. This is not important for earlier tasks in this assignment, but will certainly be a consideration if you wish to complete all tasks.

The robot is relatively small and can turn in its own radius. You have access to the drive system as a set of motor modes: turn left, turn right, drive forward, drive backward, and stop. Each new motor command overrides the previous mode, so you may not, for example, go forward and turn left at the same time. When moving, encoders in the motors provide feedback on how far the robot has driven or turned - allowing you to calculate a rough relative position. Maintaining such a position is not necessary for earlier tasks, but may prove useful on later ones. Note that when moving, the robot travels at a speed of 50 millimetres per second, and turns at a rate of 30 degrees per second.

[1] M.Maimone, P.C. Leger, and J.Biesiadeck. Overview of the mars exploration rovers autonomous mobility and vision capabilities. In *IEEE International Conference on Robotics and Automation Space Robotics*, 2007.

The platform has a final function which allows it to drop a sensor probe. You will use this in the final part of the exercise.

The assignment is divided into four tasks, each of which implements a single command the operator can send to the robot. You will most likely find it useful to reuse components from one task again in later tasks, so it might be worth bearing this in mind whilst structuring your solution.

Task 1 [10%]

Implement the *turn* command, turning the robot a number of degrees. Positive values turn right (clockwise), negative values turn left (anti-clockwise). You should report back to the operator the actual number of degrees turned, which may be a few degrees different to the specified value. You can safely assume that you won't bump into anything whilst turning.

Task 2 [30%]

Implement the *move* command, moving the robot a distance forward or backward. Positive values are forwards, negative values are backwards. You will need to monitor the hazard detectors and stop if an obstacle gets too close (the operator will then decide how to deal with the situation). If the robot stops normally (i.e. no hazards were detected) return *move.complete*, but if a hazard is detected return *hazard.detected*. In both cases you should also return the number of millimeters actually moved.

Task 3 [30%]

Implement the *find.blob* command. Taking the colour value provided, you should search for a matching blob in the 360 degrees surrounding the robot. As the object camera only has a 90 degree field of view you will need to rotate to perform the search. At the end of the search you should return the new heading of the robot relative to the heading you started at (as you will have rotated). When a match is found, you should stop the search and return *blob.found* with the blob's data structure. If multiple matches are found, you may return any of them. It is possible that no match will be found, in which case you should return *blob.not.found*.

Task 4 [30%]

Implement the *deploy.sensor* command. Along with this command you'll be sent two colour values (left and right). You need to drop the sensor probe as close as possible to the midpoint of two blobs matching the specified colours. This will involve turning the robot to align it with the blobs, moving forward without colliding with any objects and when close enough - stopping and deploy the sensor. You may also want to maintain your alignment with the blobs as you move forward. On successful deployment you return should *sensor.deployed*. If you cannot find the blobs for the drop zone then you should return *drop.zone.not.found*. To get full marks you will need to appropriately modify the response protocol such that it returns the new relative position of the robot and allows the reporting of any other error conditions encountered.