

## **System Context Overview:**

If we want to learn more about our solar system, we have to go out and explore. Today, we can send satellites into orbit around the earth, we can launch manned spacecraft to visit the International Space Station, we can send astronauts to the moon and, for exploration beyond our earth and moon environment, we can send probes to other planets. An example of the latter is the current mission to Mars which involves a pair of robotic rovers that are known as the Mars Exploration Rovers (MER). But why are we sending robotic rovers rather than sending people like we did when we explored the Moon?

The answer is that we aren't really at the point yet where we can send human beings to Mars. Different nations have sent more than 30 probes toward Mars, but less than one-third of those probes have survived the trip. Without higher odds of success, it is not prudent to replace those robotic probes with human beings. Another reason favoring robotic exploration is cost. Robots don't need complicated life support systems; they can tolerate a bumpy ride into the Martian atmosphere and they do not need to ever return to Earth. They do their work and, if all goes well, they communicate back their scientific information to ground stations on earth.

A manned mission would also provide additional engineering challenges. Astronauts need food for the trip, which is heavy and costly to launch into space. Astronauts would also like to return to earth, meaning they may need to produce fuel for the return mission from the Martian atmosphere. Nothing like this has ever been attempted, and it would take a number of test missions to prove the concept. Another big consideration is the cosmic radiation that astronauts would absorb during such a long mission, and how to block it. Much of this radiation is blocked on Earth by the Earth's magnetic field, but Mars has no protective magnetic field.

So for now, we need rovers, and we need a team of engineers who as a group will understand how to design, build, test and execute a robotic mission to Mars. We need aerospace, electrical and mechanical engineers. We need guidance, navigation and control algorithm specialists. We need communication systems engineers. We need scientists to help understand the sensors and map out the mission needs. And we need software engineers to architect the flight code so the rover can execute its mission autonomously, as communication delays between Earth and Mars make it awkward and inefficient to command a rover directly.

This system problem represents one type of assignment given to such an exploration rover team.

### Problem Introduction:

In this problem your team will design and build a robotic rover to survey a terrain. Your sample terrain will be composed of an 8 foot by 8 foot square grid of 16 squares. Squares will contain varying levels of mineral contents, with darker squares containing the most minerals. Your task will be to design a robotic rover capable of navigating the grid to collect data from each square in order to create a mineral map of the terrain.

Your rover will be transported to the grid by a delivery device that will choose the grid coordinates and set the initial alignment of the rover. The delivery device will manipulate the rover such that it is initially positioned in the center of a single square of the grid such that it is either perpendicular or parallel to the survey area. However, due to limitations of the guidance system for landing the delivery device, there is no guarantee within which square the rover will start its mission.

Once the delivery device has positioned the rover, it will communicate its initial position and North/South/East/West alignment by command, signaling that the rover should start its survey. Once the survey is complete the rover will telemeter its mission data for post processing.

A sample grid is provided below.

NORTH			
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

**Figure 1 – Sample Grid**

## **Problem Part I: Rover Software Design and Hardware (75 pts)**

Your team's objective is to design and build a rover capable of navigating through the terrain and recording data samples. A hardware design and software requirements are provided to you in this document. You must build a rover to the provided specifications and create a software design to implement the task at hand. You will implement the software design in Part II of this problem.

By **October 14<sup>th</sup>, 2004** your team will be expected to have turned in an initial software design. Appendix A of this document provides a sample software design for the first demo\_rover shown in class, to give you a sense of the level of detail expected. Appendix B of this document provides a sample test report to provide you with an outline for what is expected. If needed you may update the software design as part of Part II, but the initial design should be a complete design that reflects your initial approach to solving the problem. See your lecture notes for more information on software design.

### *Rover Hardware*

The hardware for your rover will be composed entirely of the contents of a Lego Mindstorms Robotics Invention Systems 2 kit that will be provided to each team. In addition, each team will be provided with an additional two Lego Mindstorms sensors for measuring wheel rotation. Construction instructions for the rover are available on the course website (mars\_rover.pdf).

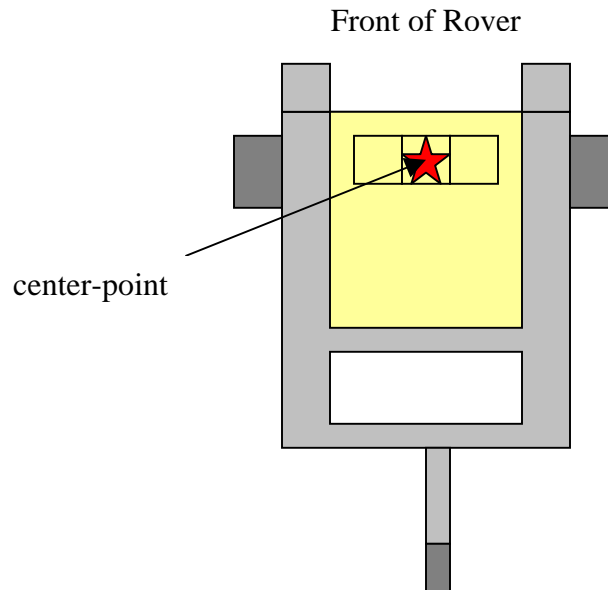
Be sure to evaluate the limitations of your hardware before proceeding to the rover software design. You should pay particular attention to the limitations of your sensors, as they are responsible for collecting mineral information and for aiding in your navigation algorithm.

Your rover will need to be able to distinguish between 4 different mineral values (ranging from white to black). Mineral samples are available in the lab. Your team should write a small piece of code to initialize the light sensor and experiment with the 'View' button on the RCX for testing the light sensor input under a variety of light sources (This piece of code is simply recommended, not required). The sensor will achieve the most accurate readings when it is as close to the ground as possible without actually touching the surface.

Familiarize yourself with the capabilities of the rotational sensor, as the rover software will need to control the rover with enough accuracy to navigate the grid. You will likely find that getting the rover to drive in a straight line and to turn in a controlled manner will require careful software calibration of the hardware. If you discover that one or both of the rotational sensors prevent the rover wheels from rotating freely, try lubricating the sensor axle with WD-40. Consider developing a small library of navigational functions, similar to those that were provided to you in the previous Lego Mindstorms problems, early in your design process. This will help you to understand the limitations of your hardware (something that will necessarily impact your software design), and it will get

you started with some of the software development and integration with the hardware before you start phase II of the project.

It is recommended, though not required, that you primarily drive the Rover with the larger wheels to the front. The larger wheels will be considered the front of the rover for the purposes of describing the rover position.



**Figure 2 – Rover Orientation**

### Rover Software

Your team will be developing the software for the rover in Ada. Ada is a programming language that was initially developed by the United States Department of Defense for embedded computer systems. For this project, you will be using the same AdaGIDE environment with the Ada/Mindstorms API that you used in the previous problem sets. You will use AdaGIDE to develop, compile, build, and upload your software onto the RCX. See Appendix C for additional tips and tricks for this problem related to the AdaGIDE environment and the RCX.

The rover software must work with the rover hardware to accomplish the required tasks. During the demonstration, your rover will be placed with the center-point in the center of an arbitrary cell on the grid. You will be provided with the coordinates and orientation of your rover via an infrared command.

Without receiving any further infrared commands, your rover should then explore the entire grid and store a mineral level for each square of the grid. This reading should be written to the datalog on the RCX for download and post-processing once the rover has completed its mission. You will not be given the initial location and orientation of the

rover when asked to post-process the data, so the rover must include in the datalog sufficient information to construct the terrain map after the mission is complete.

The specific requirements for the rover software are as follows:

1. The rover shall receive an infra-red command communicating the initial grid position and orientation.
2. The rover shall emit an auditory signal upon receiving this command.
3. The rover shall use the receipt of this command as the signal to start data collection.
4. The rover shall collect sufficient telemetry to construct a topographical map of the area investigated, with one data sample for each square of the grid.
5. The rover shall store telemetry data in the data log.
6. The rover shall telemeter the data log data via the infra-red hub when the hub requests the data (the robot and or hub can be moved to ensure the two are facing each other).
7. The center-point of the rover shall remain on the grid, where the center-point is defined as the center of the front axel (see figure 2).

Your software design should be sufficient to meet all of these requirements. Include information on global data and any interesting algorithms that you will need to solve the problem. Be sure to include details on how you plan to organize the datalog information such that you can reconstruct the grid's mineral deposits levels.

**Note: You will be responsible for developing all of the software to complete this problem. Unlike previous problem sets, you will not be provided with any starting code.**




#### Command Format

The command format for the initial positioning command to the rover will be a single 8-bit byte structured as follows:

Orientation (2 bits)	Position (6 bits)
----------------------	-------------------

The value of the Orientation bits will be 00 to indicate that the front of the rover (see figure 2) is pointing North, 01 for South, 10 for East, and 11 for West. The position bits will indicate the numeric placement on the grid. Example commands and interpretations are provided below (see Figure 3 for the associated placement on the grid).

Command (in binary)	Rover Position
00000001	Square 1 pointing North
01000110	Square 6 pointing South
11001100	Square 12 pointing West

NORTH			
	2	3	4
5		7	8
9	10	11	
13	14	15	16

**Figure 3 – Sample Rover Initial Positions**

## **Part II: Software Implementation and Test (100 pts)**

Write the rover software based on your software design. Once you believe the software is complete, you will need to test your rover to verify its ability to collect data. Your rover will only get one chance to scope out the entire demonstration grid, so it is imperative that any foreseeable issues be resolved before deployment.

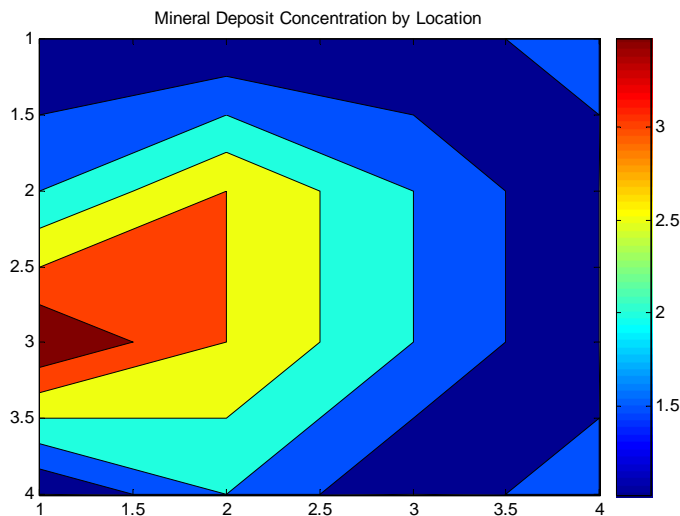
A test grid will be available in the lab with a sample mineral configuration. Be sure to test your rover with different starting positions on this grid, and examine the resulting data log.

You may find that you need to deviate from your original software design. If you do so, please update the design document so that it remains an accurate representation of your code. A copy of the completed software along with a corresponding software design document is due on **October 21<sup>st</sup>, 2004**.

### Part III: Demonstrate the Rover (25 pts)

On demonstration day, **October 21<sup>st</sup>, 2004**, you will provide your rover, preloaded with software. A TA will place the robot on the grid in the center of an arbitrary square, position the USB tower such that it can transmit to the rover, send the start command, listen for the confirmation beep from the rover, and wait for the rover to complete its traversal of the grid. Once the rover is done, the TA will remove the rover from the grid and use the USB tower to request a data dump from the rover memory. You will then be provided with the resulting data file from the rover for post-processing.

Using the data downloaded from the rover, create a contour plot in Matlab using the `contourf` command. Your code should generate a contour plot similar to the one below.



**Figure 4 – Sample Contour Plot**

You should also create a brief Test Report (1-2 pages) that should include the following:

1. A description of the tests your team performed before demonstrating the rover. (For example, was the rover tested or calibrated prior to running it in the Aero/Astro Lab? Were problems encountered in this early testing?)
2. A description of the test results. (This paragraph includes whether or not the rover worked, how the telemetry data was analyzed, etc.)
3. A discussion on the challenges faced. (Detail any problems that you encountered during the test that may have impacted the accuracy of your data. If your rover did not work, use this section to describe the approach for fixing it).

### **Grading Scheme & Summary of Things to Turn in:**

Part I: Due October 14<sup>th</sup>, 2004 (75 pts)

- Initial Software Design Document

Part II: Due October 21<sup>st</sup>, 2004 (100 pts)

- Copy of completed software
- Updated Software Design Document (to match completed software)
- Rover pre-loaded with software

Part III: Due October 21<sup>st</sup>, 2004 (25 pts)

- Raw data from rover (collected by TA)
- Contour plot interpreting data
- Test report



## Appendix A: Sample Software Design for Maze Robot Problem (from demo\_rover.adb)

The software design document should capture the critical aspects of your design. This includes, the packages used by your surveyor the global constants and variables used within your program, the headers of procedures and functions used within the program, the control flow within each of the subprograms presented in the algorithm format discussed in lectures.

### Global Data

```
Left  : constant Integer := 0; -- defines left to be 0
Right : constant Integer := 1; -- defines right to be 1

Drive : constant Output_Port := Output_A; -- sets drive to output A
Steer  : constant Output_Port := Output_B; -- sets steering to output B

-- straight input comes from sensor 1
Straight : constant Sensor_Port := Sensor_1;

-- bumper input comes from sensor 2
Bumper    : constant Sensor_Port := Sensor_2;
```

### Subprograms:

```
Header: procedure Steer_Left
Purpose: Procedure to steer the rover to the left
Preconditions: Steering motor is turned off
Inputs: none
Outputs: none
Postconditions: rover is steered towards the left, steering motor is
turned off.
Algorithm:
    Move the Rover to the left by turning steer motor on
    Wait for half a second
    Switch off the power to steering motor

Header: procedure Steer_Right
Purpose: Procedure to steer the rover to the right
Preconditions: steering motor is turned off
Inputs: none
Outputs: none
Postconditions: rover is steered towards the right, steering motor is
turned off.
Algorithm:
    Move the Rover to the right by turning steer motor on reverse
    Wait for half a second
    Switch off the power to steering motor
```

## Appendix B. Sample Test Report

The sample test report consist of three sections:

### Test Description

Name of Test	A simple name that captures the essence of what is being tested such as <i>Light_Sensor_Calibrartion</i> . Do <b>not</b> use a name like Test_1.
Functionality being Tested	One or two sentences describing what the test is being used for.
Variables affected:	List the variables that are exercised/ will be affected because the test is performed
Values used for Testing	Detail out the set of values that will be used to carry out the test

### Test Results

Name of Test	Same as the test description
Expected Behavior	One or two sentences describing the expected behavior of the system. Identify any change in variable values that are expected
Actual Behavior	Detail the actual behavior of the system in terms of variable values and observed behavior of the surveyor

### Challenges Faced

The challenges section consists of a single paragraph that details the challenges that were faced in carrying out the test. Include in this section, descriptions of design decisions that made testing easier/harder/impossible.

## Appendix C: AdaGIDE, Bricx, and the RCX -- Tips and Tricks

### Instructions for Communicating with the Rover using Bricx

To send a message to the rover, use the messaging interface of the Bricx program. First, let Bricx find your RCX by placing it in front of the USB tower when executing the Bricx program. Then find the message sending panel under Tools in the menu bar. Locate the rover to a square in the grid, and transmit the appropriate starting message from the USB tower.

Upon completion of the mission, the rover's datalog should be uploaded to the computer via Bricx. The datalog interface is provided under the Tools in the Bricx menubar.

NOTE: You cannot use AdaGIDE to communicate with the RCX while Bricx is running. If you have trouble downloading your software into the RCX from AdaGIDE, make sure Bricx is not running, disconnect and reconnect the USB tower, and try again.

### Program Size Limitations and AdaGIDE

It is possible that AdaGIDE will complain about limited memory when you try to download your software into the RCX. An explanation and work-around are provided below.

In order to download your software into the RCX, AdaGIDE first calls the ada2nqc.exe program which translates your Ada/Mindstorms code into Not Quite C (NQC) code. NQC is a variant of the C language that you may have heard of.

When you tell AdaGIDE to compile and build your Ada/Mindstorms application, a NQC file is generated in the same directory as the initial Ada source file. The NQC file can be viewed in Bricx and can also be compiled, built, and downloaded to the RCX from Bricx. Why is this important, you ask?

The Ada translator only provides a mechanism for the translation of Ada *procedures*. All of these Ada *procedures* are translated into *functions* in the NQC file. Due to how *functions* are treated in NQC, they are all inlined. This means that every time a call to a particular Ada procedure is made, the code for that procedure is essentially copied verbatim to the location of the call. This becomes a problem for us since the code for the RCX is large. However, there is a workaround.

Not Quite C also has provisions for sub-routines. A sub-routine in NQC is the equivalent of a non-inlined function. This means that your code will not be copied verbatim to the locations of its calls. Instead, subroutines allow a single copy of some code to be shared between several different callers. Of course this means the resulting program will not be as time-efficient. However, it also means we can get our rover software to run on the RCX in spite of the limited space.

In order to modify a function into a sub-routine, find its definition in the NQC code:

For example, the ‘begin’ line in Ada will look like:

```
procedure Move_East is
```

The corresponding line in the NQC file is:

```
void MOVE_EAST ( )
```

Modify this line to read:

```
sub MOVE_EAST ( )
```

You must carefully select which functions you turn into subroutines, as there are significant restrictions places on them:

- Subroutines cannot accept any arguments
- A subroutine cannot call another subroutine
- A maximum of 8 subroutines are permitted for RCX 2.0

You should consider selecting the most basic navigation functions in your application to turn into a subroutine in order to avoid the second limitation above.

Once you have modified the NQC code it will no longer be exactly equivalent to the Ada source. You will need to compile, build, and download the modified code to the RCX via Brickx.