Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
Cambridge, MA 02139

Unified Engineering
Fall 2005

System Problem #6

Due Dates: November 10,17,28-30 2005

	Time Spent (minutes)
Part I	
Part II	
Part III	
Study Time	

Name: ______________

Group Members: ____________________________

Unified SP6: System Context Overview

image credit - NASA JPL

If we want to learn more about our solar system, we must 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 recent mission to Mars which has involved 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 fewer 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 navigate between two waypoints on a terrain. Your sample terrain will be composed of a 5 foot by 5 foot square grid of 25 squares. Some squares of the grid will be passable – others will not. Your task will be design a robotic rover capable of navigating the grid to the desired endpoint, without crossing the impassible areas. Your rover must store the path traveled along the way and the terrain that it encountered at each square. Terrain will be measured using a light sensor. Dark squares are considered impassible. Your rover should not travel through two sides of a dark square, but it may enter a dark square only to determine it should not proceed. It should then back up to the last square it came from and try another route.

Your rover will be transported to the grid by a delivery device that will locate 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 square 1 of the grid, with the front of the rover facing east. The desired endpoint is square 25.

Once the survey is complete, the rover will send its mission data to earth for post processing. In this case, "earth" will be a PC and the communication link will be the IR connection between the lego rover and the PC.

Mission data from the rover will contain the light sensor reading from each square that was encountered. From the mission data, an analyst on earth should be able to derive the path traveled by the rover and the sensor readings at each square on the path. This mission information is known as telemetry data.

A sample grid is provided in figure 1.

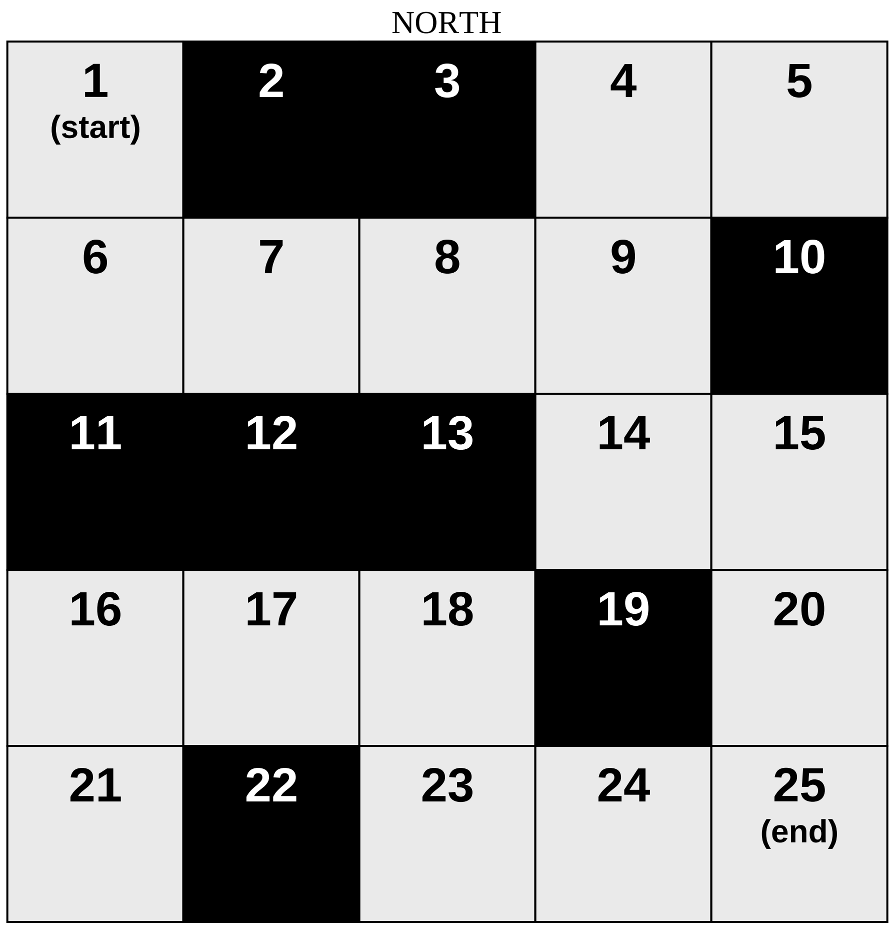

Figure 1 – Sample Grid

Problem Part I: Rover Software Design (75 pts)

Your team's objective is to design software for your current rover to make it capable of navigating through the terrain. Software requirements are provided to you in this document. You must create a software design to implement the task at hand. You will implement that design in Part II of this problem.

By **November 10, 2005** your team will be expected to have turned in an initial software design. (Appendix A of this document provides a sample software design from a similar Mars Rover that was designed to traverse a grid while reading light sensor data and storing the data in the data log.) 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

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 information about the terrain and for aiding in your navigation algorithm.

Your rover will need to be able to distinguish between 2 surface values (passable and impassable). Surface samples are available in the lab. Your team should use the light sensor input to distinguish between these values. White tiles will stand in for passable surfaces; black tiles will be those that the rover cannot cross. It will be permissible to drive into the center of black tiles to determine whether or not they are black (and doing so will simplify your navigation algorithms).

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 know already from Homework 8 that getting the rover to drive in a straight line and to turn in a controlled manner requires careful software calibration of the hardware (you should be able to reuse code from the perfect infinite square problem as part of this solution). You will need to design an algorithm that navigates the grid accurately enough for the Rover to be certain of which square it is in, which will likely involve coming up with ways to minimize the amount of driving that the rover has to do.

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.

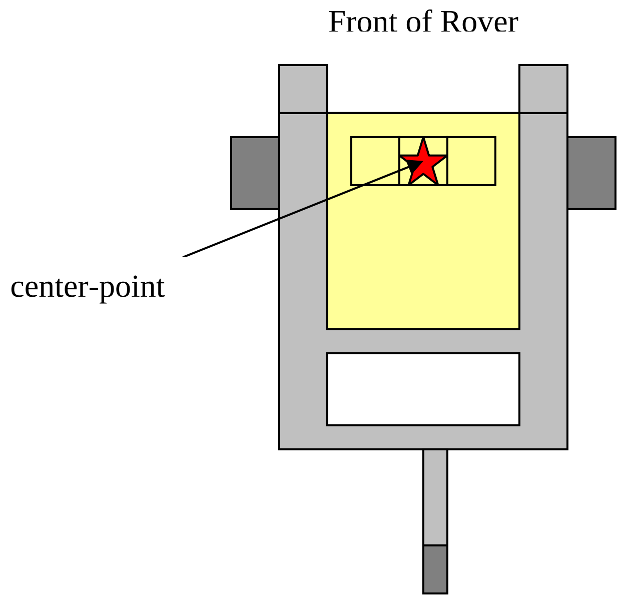

Figure 2 – Rover Orientation

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 on the previous system problems. You will use AdaGIDE to develop, compile, build, and upload your software onto the RCX. See Appendix B for additional tips and tricks for this problem related to the AdaGIDE environment and the RCX.

The specific requirements for the rover software are as follows:

1. The rover shall assume that its initial position and orientation are square one pointing east.

2. The rover shall assume that square one and square twenty-five are both passable (white).

3. The rover shall navigate to the center of square twenty-five.

4. The rover shall stop driving once in the center of square twenty-five.

5. The rover shall collect sufficient telemetry to allow reconstruction of its path through the grid, where its path through the grid is defined as the order list of passable (white) squares visited, beginning with 1 and ending with 25.

6. The rover shall store telemetry data in the data log.

7. The rover shall telemeter the data log data via the infra-red hub when the hub requests the data.

8. 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).

9. The rover shall only traverse passable (white) squares, where traverse is defined as beginning in one square, driving through a second square, and ending in a third square that is different from either of the previous two. (It is permissible to drive partway into a square to determine whether or not it is passable, but if the square is impassable, the rover needs to go back to the previous square before going anywhere else).

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 use any data structures to store collected information about the grid.

Hint: Consider using a right or left-hand rule to solve this problem. Right-hand rule algorithms attempt to keep a "wall" to the right at all times (Left-hand rule algorithms do the same thing only on the left side). By keeping a wall on one side, a robot can navigate through an unknown terrain, such as a maze. Think about what the "walls" are in this problem.

Part II: Software Implementation and Test (75 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 correctly navigate the grid. Your rover will only get one chance to scope out the demonstration grid, so it is imperative that any foreseeable issues be resolved before deployment.

A test grid will be available in the lab for configuring sample paths. Be sure to test your rover with different combinations of passable and impassable squares.

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 **November 17, 2005.**

Part III: Demonstrate the Rover & Write Test Report (50 pts)

On demonstration day, (held by team on **November 28-30, 2005**) you will provide your rover. A TA will load the software that you provided on November 17th onto the rover for demonstration. The TA will then place the robot on the grid in the center of square one 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. You will also be asked to include a copy of this data in each of your individual test reports.

Using the data downloaded from the rover, you should create a diagram explaining the path taken by the rover (also included in your individual test report). This need not be the most efficient path through the grid, but it should be a path that goes from square 1 to square 25 without including impassable squares. In the example below the solution would be **incorrect** if any of the following squares were black: 1, 2, 3, 4, 9, 13, 14, 15, 17, 18, 18, 20, 22, 23, 24, or 25 (though you may assume that 1 and 25 are both white).

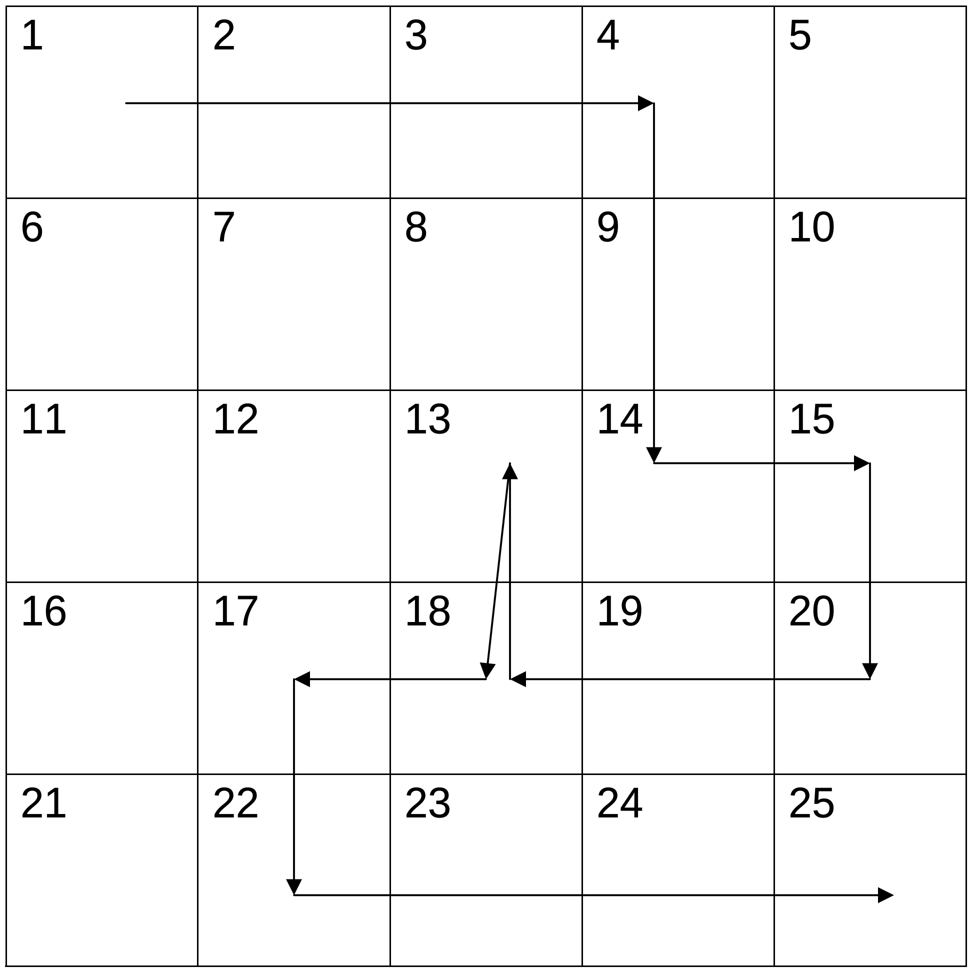

Figure 4 – Sample Rover Path

Grading Scheme & Summary of Things to Turn in:

Part I: Due November 10, 2005 (75 pts)

- Initial Software Design Document (team)

Part II: Due November 17, 2005 (75pts)

- Copy of completed software (team)
- Updated Software Design Document (to match completed software) (team)

Part III: Due November 28-30, 2005 (50 pts)

- Rover (team) – TA will download software that was provided on 11/17
- Raw data from rover (collected by TA and provided to team)
- Path reconstruction based on rover data (individual work to include in report)
- 3-4 page test report (individual) (template available for download on website)

Appendix A: Sample Software Design for Unified Terrain Sampling Rover Problem (Fall 2004)

This rover was designed to traverse all of the squares in a 4x4 grid of tiles. The rover should take a sensor reading of each square. Once the rover has covered all of the squares, it should send the sensor readings to "earth" (i.e. the PC) via the USB port on the lego RCX.

Sample Software Design

Global Data

```
Constant Int GridSize = 4
Constant Int N = 1
Constant Int S = 2
Constant Int E = 3
Constant Int W = 4

Int Location
Int Orientation
```

Methods

`Main`: Calls `Configure, Initialize`, `GoHome`, and `InvestigateTerrain`, in that order.

`Configure`: Creates datalog and configures sensors and motors

`Initialize`: Accepts the command, decodes and validates the starting location and orientation and beeps

`GoHome`: Uses the position and orientation data to navigate the rover back to square 1

`InvestigateTerrain`: Navigates through the grid, collecting mineral level and location information for the data log for each square. Psuedo-code follows below:

```
Loop 2
     InvestigateOneRow(true,GridSize)
     MoveSouth
     InvestigateOneRow(false,GridSize)
     MoveSouth
End Loop
```

`InvestigateOneRow(boolean goEast, int RowSize):` Samples data from `RowSize` squares in the direction specified by the `goEast` parameter (this method investigates to the east if this parameter is true, otherwise it investigates to the west). Psuedo-code follows below:

```
     Loop RowSize - 1 times
          Add Location & current mineral reading to data
log
          If(goEast)
               MoveEast
          Else
               MoveWest
          End if
     End loop
     Add Location & current mineral reading to data log
```

`MoveEast`: Regardless of orientation, move the rover East 1 square, update `Orientation` and `Location`. (Does nothing if moving the rover would cause it to go off the terrain.)

`MoveWest`: Regardless of orientation, move the rover West 1 square, update `Orientation` and `Location`. (Does nothing if moving the rover would cause it to go off the terrain.)

`MoveNorth`: Regardless of orientation, move the rover North 1 square, update `Orientation` and `Location`. (Does nothing if moving the rover would cause it to go off the terrain.)

`MoveSouth`: Regardless of orientation, move the rover South 1 square, update `Orientation` and `Location`. (Does nothing if moving the rover would cause it to go off the terrain.)

`Right90`: Turns the rover to the right by 90 degrees and stops

`Left90`: Turns the rover to the left by 90 degrees and stops

`ForwardOne`: Drives the rover straight one cell and stops

Appendix B: AdaGIDE, Bricx, and the RCX -- Tips and Tricks

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 Bricx.

SOFTWARE TEST REPORT

for the

MIT Unified Computers & Programming
Mars Rover System Problem

28 November 2005

Prepared for:

Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
Fall 2005

Prepared by:

[Name]
[Campus Contact Information]
[Lego Team #]

1. EXECUTIVE SUMMARY

1.1 System Overview

This paragraph shall briefly state the purpose of the system and the software to which this document applies. It shall describe the general nature of the system and software; summarize the history of system development, operation, and maintenance. Take a portion of the project writeup and include it here.

1.2 System Architecture & Software Components

This paragraph shall identify the parts of the system (i.e. hardware and software) that were either given or developed as part of the problem. Include how you created your software code (for example, how did you use your solutions to the psets?)

2. TEST ENVIRONMENT

2.1 Software Items Under Test

This paragraph shall identify any software by name, number and version, as applicable <u>that was tested</u> and summarized in this report.

2.2 Components in the Software Test Environment

This paragraph shall identify by name, number, and version, as applicable, the support software items (e.g., operating systems, compilers, communications software, related applications software, etc.) necessary to <u>perform</u> the testing activities on the software identified in paragraph 2.1.

3. TEST RESULTS

3.1 Overall Assessment of the Software Tested

This paragraph shall:

(a) Provide an overall assessment of the software as demonstrated by the test results in this report

(b) Identify any remaining deficiencies, limitations, or constraints that were detected by the testing performed.

3.2 Detailed Test Results

3.2.1 Rover Maneuver Test(s)

This paragraph shall describe the tests performed and the challenges faced when checking the rover's ability to move forward, backward and turn.

3.2.2 Sensor Data Test(s)

This paragraph shall describe tests performed to check the rover's reading of the light sensor data.

3.2.3 Mission Data Transmit Test(s)

This paragraph shall describe the tests used to check that the mission data could be transmitted to "earth.".

3.2.4 System Test(s)

Summarize what your rover did the first time you tried it on the 5x5 grid. Did you have to correct your algorithm after the first test? How did you learn from the rover's behavior?

How did your rover perform during the final demonstration for credit? What was its path across the grid? Include a copy of your team's output from the rover during the demo..