Dear 16.070 Student,

Congratulations, you have made it through programming and hardware to the final project! The 16.070 Spring 2002 Final Project is to design, develop, test and demonstrate an autonomous satellite docking with the International Space Station (ISS). This docking scenario will be built up from code that you have already developed in your problem sets.

The project is divided into four parts, each due separately (see Figure 1). In the first part, you will create a handyboard-based automatic satellite docking controller. For the second part of the project, you will use the handyboard controller from the first part to control a workstation based simulated satellite through serial line. The third part of the project will be to integrate the handyboard controller from the first part into a hardware simulation that will be provided to you. The last part of the project is a test report that will give you a chance to demonstrate that your end product meets the specifications contained in this document.

The parts of the final project are expected to be challenging. Creating all required software from scratch would be a daunting prospect, so you are encouraged to use software you have written previously in 16.070. As a second, but less encouraged option, you may use code in the problem set solutions in your final project code. Beyond that, usual collaboration rules apply.

Due to the culminating and time consuming nature of this project, its weight on your overall grade will be significant. Read through the entire project thoroughly before beginning. Start assignments early, ask questions often, follow instructions carefully, and TEST TEST TEST!!!

Good luck,
16.070 Team

---

**Figure 1 - Complete Final Project**

The parts of the final project are expected to be challenging. Creating all required software from scratch would be a daunting prospect, so you are encouraged to use software you have written previously in 16.070. As a second, but less encouraged option, you may use code in the problem set solutions in your final project code. Beyond that, usual collaboration rules apply.

Due to the culminating and time consuming nature of this project, its weight on your overall grade will be significant. Read through the entire project thoroughly before beginning. Start assignments early, ask questions often, follow instructions carefully, and TEST TEST TEST!!!

Good luck,
16.070 Team
Part A (Due: 4/22/02)

In part A, students will implement an automatic controller on their Handyboards and a workstation-based test harness (see Figure 2). The controller will use a similar state transition diagram to that found in PS3 (see Figure 3).

In addition, the controller will input telemetry and output commands using the byte encoding/decoding specification disclosed in PS6 (see Figure 4). The Handyboard-based controller will communicate with the workstation-based test harness via a serial connection using the serial libraries introduced in Recitation 9. The test harness will issue a series of test bytes and record those test bytes and the controller’s responses in a data file. Telemetry should be issued in sequences according to expected responses determined from the state chart in Figure 3.

State Transition Diagram for HB Controller

Figure 3 – State Transition Diagram for HB Controller
Although it is not required for Part A, it is strongly recommended that additional testing be performed during the creation of the serial interface. Some recommendations on testing approaches can be found in Part D. Remember to read through the entire project description before beginning.

![Diagram of serial transfer specification](image)

**Figure 4 – Bit Specification for Serial Transfer**

In Figure 3, it is indicated that the control will not begin until the “Start button” has been pressed. Pressing the start button on the controller Handyboard will cause a byte with the start bit (bit #6) turned on to be sent over the serial port, at which point the simulation will begin. In all cases, a byte sent from the controller will be answered with a byte returned by the simulation engine (whether that is the test harness in Part A, the state based simulator in part B, or the hardware platform in part C). This means that no two consecutive bytes will ever be sent in the same direction over the serial line.

The test harness being designed in Part A will begin operating when it receives a byte from the Handyboard with only the start bit set to “1” (an integer value of 64). After receiving the start bit, the harness should begin by sending a byte of telemetry. The controller will respond with a command. This process will continue until all sample telemetry bytes have been sent and all the Handyboard’s responses to those bytes have been received and recorded. The responses from the controller will be determined by the controller’s current state and most recent input.

The nominal speed (used in the state transition diagram) will be a value of 13 in the velocity bits. This nominal speed corresponds to a telemetry byte of the form “01101???” where the ? bits are unrelated to velocity. The state transition diagram specifies that when the nominal speed is reached, the controller will no longer issue forward thrust commands. You will not need the parity bit, resend bit, or stop bit in Part A. Tables 1 and 2 elaborate on the divisions with the serial transfer specification. After sending the start byte and receiving the first telemetry byte, the Handyboard should display the ascii values of the most recent telemetry byte it received and of the command byte it issues in response.

**Note:** It is absolutely critical that the serial transfer be used exactly as it is specified in order to successfully interface with hardware provided in part C.
Table 1 – Telemetry Specification

<table>
<thead>
<tr>
<th>Telemetry Type</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity</td>
<td>7-3</td>
<td>This data is a measure of the velocity of the satellite. It uses a signed magnitude representation, with 31 equally divided ranges of velocity.</td>
</tr>
<tr>
<td>IR</td>
<td>2-1</td>
<td>The IR bits will indicate the satellite is observing one of three conditions: 1) an IR beacon is visible, 2) an IR beacon is NOT visible, 3) the satellite is docked</td>
</tr>
<tr>
<td>Parity</td>
<td>0</td>
<td>The parity bit will not be used in part A, but in part B, the parity bit will be set for each outgoing byte of telemetry by the workstation-based simulator. An uncorrupted byte will have even parity. In part C, the parity bit will be set by the onboard Handyboard.</td>
</tr>
</tbody>
</table>

Table 2 – Command Specification

<table>
<thead>
<tr>
<th>Command Type</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>7</td>
<td>The stop command (stop bit set to “1”, all others “0”) should be sent if the when the state chart reaches the final state. This can occur when the IR telemetry indicates the satellite has docked, when time has run out, or when the stop button is pressed on the controller.</td>
</tr>
<tr>
<td>Start</td>
<td>6</td>
<td>The start command (start bit set to “1”, all others “0”) should only be sent once, at the start of a simulation. After the start bit is sent from the controller, the simulation will begin.</td>
</tr>
<tr>
<td>Resend</td>
<td>5</td>
<td>This bit will not be needed in Part A, but should be used in parts B and C when a telemetry byte with incorrect (odd) parity is received. In the event of a resend command, the simulator will respond by sending the last telemetry byte again. The state and the time step in the simulator will not propagate in the event of a resend command.</td>
</tr>
<tr>
<td>Unused</td>
<td>4</td>
<td>This bit is unused and serves no purpose in the final project.</td>
</tr>
<tr>
<td>Angle</td>
<td>3-2</td>
<td>The angle bits can indicate one of three possible rotation commands: do not rotate, rotate counter-clockwise, rotate clockwise. The docking algorithm provided in the state transition diagram does not make use of clockwise rotation, however, student-altered algorithms in part C may.</td>
</tr>
<tr>
<td>Thrust</td>
<td>1-0</td>
<td>The thrust bits can indicate one of three possible thrust commands: no thrust, thrust forward, thrust in reverse. The no thrust command will be sent if the craft is rotating or has reached its nominal speed.</td>
</tr>
</tbody>
</table>

The following sequences of 8 bit integers will be tested:

<table>
<thead>
<tr>
<th>Test Value (to Controller)</th>
<th>Expected Command</th>
<th>Test Value (to Controller)</th>
<th>Expected Command</th>
<th>Test Value (to Controller)</th>
<th>Expected Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>106</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>28</td>
<td>128</td>
</tr>
</tbody>
</table>

In addition to the three test cases listed above, create three more, calculate their expected command responses, and test them using the test harness and controller.

**Turn in:** Copy your output file and your code into the 16.070HW/FinalProject/PartA directory. Turn in a paper copy of your test case choices and an explanation of the results.
Part B (Due: 5/1/02)

In this portion of the project you will create a simulation of the satellite docking vehicle that will be controlled using the Handyboard-based controller created in Part A.

![Controller WB Serial I/O controller WB WS simulator Vehicle Serial I/O On Board HB](image)

Figure 5 – Part B

The satellite simulation will use the same 2D state propagation matrix that was designed in PS7.

**State Propagation Matrix**  
\[ X_{n+1} = AX_n + Bu \]

\[
\begin{bmatrix}
X_{1_{n+1}} \\
X_{2_{n+1}} \\
X_{3_{n+1}} \\
X_{4_{n+1}} \\
\Delta \theta_{n+1}
\end{bmatrix}
= 
\begin{bmatrix}
1 & \Delta t & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & \Delta t \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X_{1_n} \\
X_{2_n} \\
X_{3_n} \\
X_{4_n}
\end{bmatrix}
+ 
\begin{bmatrix}
0 \\
\frac{\Delta m \cos \theta}{T} \\
0 \\
\frac{\Delta m \sin \theta}{B}
\end{bmatrix}
\]

*note: B must be recomputed each time theta changes.*

\( X_1 \) is x position  \( X_2 \) is x velocity  \( X_3 \) is y position  \( X_4 \) is y velocity  
\( m \) is mass  \( T \) is thrust  \( \theta \) is orientation angle [0,360]

Your simulator must encode and issue telemetry to the controller (running on a handyboard) via a serial line and using the workstation serial libraries introduced in Recitation 9 and the bit specification in Part A. The simulator must encode a parity bit in each byte sent to the controller. In the event that the controller receives a byte that does not pass the parity test, it should send a “resend” command. The simulator should respond by resending the most recent byte of telemetry instead of propagating to the next time step.

The simulator will use the algorithm in PS2 to determine whether an infrared beacon is visible to the satellite each time telemetry is sent to the controller.

The simulation will end when the satellite has docked with the beacon, the controller sends a stop command (activated by one of the end-state conditions on the state transition diagram), or the satellite travels outside the bounds of the simulation space (defined below).

The simulation will use a graphics package located at [http://web.mit.edu/graphics](http://web.mit.edu/graphics), the same one that was used in PS7. Instructions for using the graphics package files are in located in the directory as well.

In addition to outputting graphics to the screen, the simulator will also write a telemetry file containing the following information on each line:

<time> <x pos> <x vel> <y pos> <y vel> <angle> <thrust command> <rotation command>
Your simulator should operate using an algorithm similar to the following pseudocode:

```plaintext
put initial conditions & declarations here
WHILE (not_exit)
    read data from Handyboard (serial in)
    calculate physics (vehicle altitude/velocity computation)
    write telemetry to file
    write to Handyboard (serial out)
    call graphics package
END WHILE
```

**Simulation Conditions:**

The virtual space inside which the simulation will be run is a grid from –20 meters to +20 meters in the x and y directions.

The starting position of your satellite for all test cases will be (-5, -5).

The position of the beacon in all test cases will be (5, 5).

Your satellite will be considered docked if it reaches a position within 0.1 meters of the beacon.

Your simulator time step will be 0.2 seconds.

The mass of the satellite is 5 kg.

When the thrust is off, no forces act upon the satellite.

The satellite may only rotate when it is at rest and not thrusting.

The satellite may only thrust when it is not rotating.

Thrust force is 20 Newtons.

Rotation rate is 5 degrees per second.

The IR receiver on your satellite will have a visibility cone of $\pm \delta = \pm 2\degree$ degrees (different from PS2).

![Figure 6 – IR Visibility Cone Geometry](image)

**Turn in:** Copy your output file, your Controller code (with any modifications you may have made since part A), and your simulator code into the 16.070HW/FinalProject/PartB directory. Also, there will be a graded demonstration of your controller and simulation in lab in Week 14.
Part C (Due: 5/8/02)

At this point, a Handyboard-based controller should have been written and tested and evaluated with a workstation-based simulation in Parts A & B. The next step is to integrate your controller onto a hardware platform that will be provided to you. The hardware will be a small vehicle that will respond like a satellite in space and an IR emitter that will represent the ISS docking beacon. The vehicle will have an on-board Handyboard (also provided) running software that will control the vehicle’s movement and interpret IR signals. The on-board Handyboard will communicate with your Handyboard (which will be mounted to the vehicle) via a serial link. The serial communication between your Handyboard running controller software and the on-board Handyboard will use the same bit specification used in Parts A and B. The on-board Handyboard will provide all the same telemetry that your simulation provided, however the simulated infrared beacon will be real and the vehicle will attempt to dock with it. A vehicle is considered “docked” when its forward microswitch assembly is activated. When one of the microswitches is depressed, the on-board Handyboard will send a telemetry byte indicating an IR value of “docked”.

The vehicle /on-board handyboard packages will be made available for check-out at the Aero/Astro Library on the 1st floor of building 33. Checkout of this equipment will be conducted according to the reserve book rules. There are 6 such packages and they can be checked out for two hours at a time.

You may tweak your controller or modify its algorithm to ensure better “real world” results. You may find that your simulator was not a completely accurate representation of the vehicle’s behavior. Given the limited amount of time you will have to test with hardware before the demonstration, it is recommended that you carefully consider and test any modifications before trying them on the vehicles.

IMPORTANT NOTE: The cable distributed for connecting your controller handyboard to the onboard handyboard is NOT the same as the cable you have been using to connect the handyboard to the interface board. **DO NOT** use your handyboard/interface board cable (looks like a phone cord) to connect two handyboards. It will result in two damaged handyboards and possibly a fire. Only use the connecting cord that is issued with the vehicle / on-board handyboard package.

**Turn in:** Copy your Controller code (with any modifications you may have made since part B) into the 16.070HW/FinalProject/PartC directory. Also, there will be a graded demonstration of your controller docking the car in lab in week 15, which will be held in the hangar in building 33.
Part D (All 3 Sections Due: 5/15/02)

Section I – Final Project Test Report

Test identification – Planned tests
Complete the following 6 paragraphs outlining your proposed test plan for the entire project. The test plan should be designed to facilitate integration of the different modules and code segments of which your program is comprised and should test and verify the expected outputs/results of the different modules.

Note: This test plan (if begun early) will be very helpful for you to integrate and debug your final project. It is highly recommended that you keep a lab notebook to record the results from tests that you run while implementing parts A through D. Although some parts of the Test Report cannot be written before the lectures on testing, many parts can and it is advisable to begin testing early in the project.

I.1 Serial I/O Test(s)
This paragraph shall describe the tests to be run to confirm operation of the serial I/O portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

I.2 Thrust/Rotation Controller Test(s)
This paragraph shall describe the tests to be run to confirm operation of the thrust controller portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

I.3 Simulator Test(s)
This paragraph shall describe the tests to be run to confirm operation of the simulation portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

I.5 Graphics Display Test(s)
This paragraph shall describe the tests to be run to confirm operation of the graphics and telemetry display portion of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

I.4 Hardware Integration Test(s)
This paragraph shall describe the tests to be run to confirm successful integration of the controller into the car/on-board handyboard package in part C of the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

I.6 System Test(s)
This paragraph shall describe the tests to be run to confirm operation of the integrated system in a docking scenario (actual conditions) for the MIT16.070 project. It shall include a description of any test code that will support the test (but is not part of the actual project software).

In addition to the tests enumerated above, the test report should include:

• an Executive Summary of the entire project
• a description of the particular aspects of the project being tested
• a description of equipment (hardware and software) used to perform testing
• documentation showing project requirements were met
• suggestions for future improvements on the project
• a detailed discussion of the results (including possible reasons for failed tests)

Section I Turn In: typed Final Project Test Report.
Section II – Failure Reports

You will find links to three failure reports on the course website, accessible from the LINKS subsection.

a) Describe two software errors from any of the failure reports. Use programming terminology.

b) Explain what you could do to your project code avoid failures similar to those that occurred as a result of these software errors.

c) What underlying premises are common to most, if not all, of the software-related failures? Identify two.

Section II Turn In: Typed answers to a, b and c.

Section III – End of the Year!

Course Survey
This is a follow-up on the course survey that you needed to complete for Problem Set 1. Please complete the course survey at http://web.mit.edu/16.070/www/survey.html. (It is nearly identical to the one you completed at the start of the semester.)

Course Evaluation
In addition to the course survey, please complete the Course Evaluation. The URL and all suitable passwords will be provided to you as soon as possible. This information was not yet available upon posting of the problem set.