16.851 - SATELLITE ENGINEERING MEMORANDUM

TO: 16.851 FACULTY

FROM: STUDENTS

SUBJECT: PROBLEM SET #1

DATE: 9/17/2003

Motivation:

Launch vehicle selection and deployment strategies are an important and expensive issue for satellite development. We want to provide ease of launch vehicle selection given spacecraft design specifications.

Problem Statement:

Design a launch vehicle decision tool. Given specific payload constraints, such as payload mass, payload dimensions, required launch frequency, maximum acceleration loads, and desired orbit, output the launch vehicles that satisfy those requirements. The design tool then should be able to optimize the selection on other criteria such as cost, risk, performance margin, and availability. The tool will be capable of being updated with current launch vehicle specifications as they are available (reliability, cost, availability, payload mass, etc.)

Approach:

First, a literature survey will be performed to gather currently available launch vehicle specifications. This data will be stored in a standardized data structure for ease of future updates. Next, a selection flow chart will be laid out to prioritize down selection criteria/logic for the software tool. The software tool will then be coded incorporating the data and selection logic. The tool will allow the user to control the number of input constraints.

Solution:

I. Requirements Specifications:

Given specific payload constraints (Table 1), output the launch vehicles that satisfy those requirements and optimize them according to user flagged options. Any combination of constraints can be applied to the selection process. The tool should be coded generically so that any new or updated launch vehicle can be added to the database as manufacturer data are available. Given a databases of launch vehicles, the tool will expand the database to include all variants of any one class of vehicle. After down selecting a set of vehicles by applying the constraints, the tool will optimize the remaining set of vehicles by a particular parameter (Table 2).

II. Problem Inputs:

The program inputs consist of a payload data structure containing the desired constraints and the optimization flags (Table 1 & 2). This structure is generated by the "payload_dat.m" file (please see attached file) The launch vehicle database consists of a file for each vehicle class that contains all of the relevant specification data and one central file that loads all the vehicle data plus the payload data ("load_data.m"). The launch vehicle data also includes information on the vehicle class variants and the appropriate launch sites for the vehicle. (Please see attached files). The supplied database contains many common vehicles used for commercial payload launches.

Table 1: Payload Constraints for Launch Vehicle Tool

Constraint	Description
mass	[kg], total payload mass (spacecraft, adaptors, propellant, etc)
diameter	[m], payload diameter
height	[m], payload height
inclination	[deg], desired orbit inclination
launchfreq	[launches/year], launch frequency
max_axial_accel	[g], maximum axial acceleration
max_lat_accel	[g], maximum lateral acceleration
lat_freq	[Hz], minimum lateral frequency
long_freq	[Hz], minimum longitudinal frequency
acoustic	[dB], noise intensity
max_aeroheating	[W/m^2], aero heating after fairing deploy
air_clean	[class], air cleanliness class
orbital_accu_alt	[km], allowable LEO insertion altitude error (+/-)
orbital_accu_incl	[deg], allowable LEO insertion inclination error (+/-)

Table 2: Payload Optimization Flags for Launch Vehicle Tool

Optimization Parameters	Description
cost	[Mil $], minimize cost
risk	[Mil $], minimize risk, $risk = (1 - vehicle\_reliability) \times vehicle\_cost$
performance margin	[kg], maximize performance margin, $pmargin = vehicle\_payload\_capability - payload\_mass$
availability	maximize availability, $availability = 1 - \left[\frac{(launchfreq) \times (1 - vehicle\_reliablility) \times (standown\_time)}{1 - 1/(surge\_capacity)} \right]$
absolute cost	[Mil $], minimize absolute cost, $absolute\_cost = vehicle\_cost + \left((1 - vehicle\_reliability) \times vehicle\_cost \right)$

III. Software Execution Procedure:

The launch vehicle selection tool is executed using Matlab. First the payload input is specified by the user by editing the "payload_dat.m" file. Next the vehicle database and payload input must be loaded into the Matlab workspace. This is accomplished by executing the script, "load_data.m". The tool is executed by entering at the Matlab command prompt:

```
>> selected_vehicles = launcher_select(payload, veh);
```

The constrained and optimized set of launch vehicles will be stored as a data structure in the "`selected_vehicles`" variable. This structure contains all the vehicle specifications plus the newly calculated optimization parameters.

IV. Expected Outputs:

The outputs consist of a sorted list of launch vehicles that meet all user input constraints and are optimized with respect to one specified user flag (Table 2). The output can also include several figures that depict various trends of the selected launch vehicles.

V. Pseudo-Code (Very Simplified):

```
Solution launch_select( payloadInput, launchVehicleDatabase) {
    Solution solution;
    Sites sites;
    chooseSites(payloadInput, launchSites);
    filterConstraints(payloadInput, launchVehicleDatabase);
    solution = sort(launchVehicleDatabase, option);
    return solution
}
```

VI. User's Guide

Hello. Welcome to LaunchSelectTool1.0!

How to select a launch vehicle:

1. Type payload information into "payload_dat.m". If you don't want one of the constraints or don't know the information, please type -1.

2. Please choose an optimization option for your output under the section named "Program Optimization Sorting Options" in "payload_dat.m".

3. Save "payload_dat.m"

4. Load the input data and vehicle database into memory by executing "load_data.m".

   ```
   >> load_data
   ```

5. Execute the launch select tool

   ```
   >> selected_vehicles = launcher_select(payload, veh);
   ```

6. All applicable vehicles will be output to the "`selected_vehicles`" data structure in optimal order according to the user option.

How to add to database:

1. Make a copy of any vehicle data file "*launch_vehicle*.m"

2. Modify the data to match the new vehicle

3. Add the new data file name to the "load_data.m" launch vehicle list.

VII. Sample Test Run

For the test run, a payload with constraints shown in Table 3 is submitted to the launcher selection tool. Table 4 shows the results sorted by cost. The negative availability values are because the current vehicle database shows those vehicles as having never launched any payload. This also has the effect of driving up the risk and absolute cost for those particular vehicles. The Atlas II's zero risk is due to its perfect launch record.

Table 3: Test Run Payload Constraints

Constraint	Value
mass	6000 kg
diameter	3 m
height	3 m
inclination	34 deg
launchfreq	2 per year
max_axial_accel	8.5 g
max_lat_accel	2.2 g
lat_freq	45 Hz
long_freq	30 Hz
acoustic	200 dB
max_aeroheating	1200 W/m^2
air_clean	10,000 class
orbital_accu_alt	20 km
orbital_accu_incl	2 deg

Table 4: Test Run Results

Vehicle	Cost (mil $)	Risk (mil $)	Performance Margin (kg)	Availability (%)	Absolute Cost (mil $)
Soyuz_U	50	1.5	220	86.1	51.5
Atlas V_400	90	90	6500	-360	180
Atlas II_AS	105	0	2618	100	105
Atlas III_A	105	105	2640	-360	210
Atlas III_B	105	105	4718	-360	210
Atlas V_500	110	110	14050	-360	220
Proton_M	112	112	15000	-360	224

VIII. References

Isakowitz, Steven J. 1999, *AIAA International Reference Guide to Space Launch Systems (3rd Edition)*. Washington, DC: American Institute of Aeronautics and Astronautics.

Jilla, Cyrus D. 2002, *A Multiobjective, Multidisciplinary Design Optimization Methodology for the Conceptual Design of Distributed Satellite Systems*. Massachusetts Institute of Technology, Aero/Astro.

Wertz, James R. and Wiley J. Larson (Ed.), *Space Mission Analysis and Design*. El Segundo: Microcosm Press, 1999.

IX. Appendix - Source Code Listing

Payload Constraints & Optimization Flags, *payload_dat.m*

```
%##################################
%## payload data

%specify a "-1" value to disable constraint

payload.name = 'Test Payload';
payload.mass = 6000; %[kg], total payload mass (spacecraft, adaptors, propellant)
payload.diameter = 3; %[m], payload diameter
payload.height = 3; %[m], payload height
payload.orbit_type = 'leo'; %orbit class // options: [leo, polar, sunsync, ss, gto, geo]
payload.inclination = 34; %[deg], desired orbit inclination
payload.launchfreq = 2; %[launches/year], launch frequency
payload.max_axial_accel = 8.5; %[g], max axial acceleration
payload.max_lat_accel = 2.2; %[g], max lateral acceleration
payload.lat_freq = 45; %[Hz], min lateral frequency
payload.long_freq = 30; %[Hz], min longitudinal frequency
payload.acoustic = 200; %[dB], noise intensity
payload.max_aeroheating = 1200; %[W/m^2], aeroheating after fairing deploy
payload.air_clean = 10000; %[class], air cleanliness class
payload.orbital_accu_alt = 20; %[km], allowable LEO insertion altitude error (+/-)
payload.orbital_accu_incl = 2; %[deg], allowable LEO insertion inclination error (+/-)

%##################################
%## Program Optimization Sorting Options

% 1 for category sort on // 0 for category sort off
% You may only enable one sort category at a time.

payload.cost = 0; %minimize cost
payload.risk = 0; %minimize risk
payload.pmargin = 0; %maximize margin
payload.availiability = 0; %maximize availability
payload.absolute_cost = 1; %minimize absolute cost
```

Load Data File, *load_data.m*

```
%##################################
%## load data

clear
clc

%loads payload data
payload_dat

%Loads all launch vehicles

i=0;

Ariane4
Ariane5
AtlasII
AtlasIII
AtlasV
DeltaII
DeltaIV
Molniya
Pegasus
Proton_a
Proton_b
Soyuz

clear i j
```

Sample Vehicle Data File, *AtlasII.m*

```
% Launch Vehicle Database

i=i+1;

veh{i}.family = 'Atlas';
veh{i}.class = 'Atlas II';
veh{i}.country = 'USA';
veh{i}.provider = 'Lockheed Martin';
veh{i}.success_flight = 41; %# of flights%
veh{i}.total_flights = 41;
veh{i}.stddwntime = 0.3; %years
veh{i}.surge = 1.15; %percentage
veh{i}.max_axial_accel = 6; %g
veh{i}.max_lat_accel = 2; %g
veh{i}.min_lat_freq = 10; %Hz
veh{i}.min_long_freq = 15; %Hz
veh{i}.shock = 4000; %g
veh{i}.acoustic = 131; %dB
veh{i}.fairing_press = 7; %kPa/s
veh{i}.max_aeroheating = 1135; %W/m^2
veh{i}.air_clean = 100000; %class
veh{i}.orbital_accu_alt = 19.4; %km
veh{i}.orbital_accu_incl = 0.02; %deg
veh{i}.rate = 7; %# per year

veh{i}.site{1}.name = 'KSC';
veh{i}.site{1}.min_incl = 28.5; %deg
veh{i}.site{1}.max_incl = 55; %deg

veh{i}.site{2}.name = 'Vandenberg';
veh{i}.site{2}.min_incl = 63; %deg
veh{i}.site{2}.max_incl = 120; %deg

  %-- Variants / Upper Stages --
  j=1;
        veh{i}.upper_stage{j}.var_name = 'A';
        veh{i}.upper_stage{j}.mass2leo = 7316; %kg
        veh{i}.upper_stage{j}.mass2polar = 6192; %kg
        veh{i}.upper_stage{j}.mass2SS = 6074; %kg
        veh{i}.upper_stage{j}.mass2sunsync = 0; %kg
        veh{i}.upper_stage{j}.mass2gto = 3066; %kg
        veh{i}.upper_stage{j}.mass2geo = 0; %kg
        veh{i}.upper_stage{j}.cost = 85; %million dollars
        veh{i}.upper_stage{j}.fairingheight = 4.1; %m
        veh{i}.upper_stage{j}.fairingdiameter = 2.9; %m

  j=j+1;
        veh{i}.upper_stage{j}.var_name = 'AS';
        veh{i}.upper_stage{j}.mass2leo = 8618; %kg
        veh{i}.upper_stage{j}.mass2polar = 7212; %kg
        veh{i}.upper_stage{j}.mass2SS = 7225; %kg
        veh{i}.upper_stage{j}.mass2sunsync = 0; %kg
        veh{i}.upper_stage{j}.mass2gto = 3719; %kg
        veh{i}.upper_stage{j}.mass2geo = 0; %kg
        veh{i}.upper_stage{j}.cost = 105; %million dollars
        veh{i}.upper_stage{j}.fairingheight = 4.1; %m
        veh{i}.upper_stage{j}.fairingdiameter = 3.7; %m
```

Launch Select Tool, *launcher_select.m*

```
function bestveh = launcher_select(payload, veh_input);

% This is a launch vehicle selection tool
% input #1 - payload specifictions
% input #2 - launch vehicle specs


%expand database for upper stages
kk = 1;

for ii = [1:length(veh_input)]

  %## if generic vehicle can lift payload, add to database

  for jj = [1:length(veh_input{ii}.upper_stage)]
    veh{kk} = veh_input{ii};
    veh{kk}.class = [veh_input{ii}.class '_' veh_input{ii}.upper_stage{jj}.var_name];

                veh{kk}.mass2leo = veh_input{ii}.upper_stage{jj}.mass2leo;
                veh{kk}.mass2polar = veh_input{ii}.upper_stage{jj}.mass2polar;
                veh{kk}.mass2ss = veh_input{ii}.upper_stage{jj}.mass2SS;
                veh{kk}.mass2sunsync = veh_input{ii}.upper_stage{jj}.mass2sunsync;
                veh{kk}.mass2gto = veh_input{ii}.upper_stage{jj}.mass2gto;
                veh{kk}.mass2geo = veh_input{ii}.upper_stage{jj}.mass2geo;
                veh{kk}.cost = veh_input{ii}.upper_stage{jj}.cost;
                veh{kk}.fairingheight = veh_input{ii}.upper_stage{jj}.fairingheight;
                veh{kk}.fairingdiameter = veh_input{ii}.upper_stage{jj}.fairingdiameter;

        veh{kk} = rmfield(veh{kk}, 'upper_stage');
                kk = kk+1;
  end


end


% filter based on all constraints
site_found = 0; %flag found
m = 1; %counter for qualified vehicles
for i = [1:length(veh)]
    %site and inclination constraint
    for j = [1:length(veh{i}.site)]
        if(veh{i}.site{j}.max_incl >= payload.inclination & veh{i}.site{j}.min_incl <= payload.inclination)
            site_found = 1;
        end
    end

    if(site_found == 0)
        continue;
    end

    %mass constraint
    if(strcmp(payload.orbit_type,'leo') == 1)
        if(veh{i}.mass2leo < payload.mass)
            continue;
        end
    end
    if(strcmp(payload.orbit_type,'gto') == 1)
        if(veh{i}.mass2gto < payload.mass)
            continue;
        end
    end
    if(strcmp(payload.orbit_type,'geo') == 1)
        if(veh{i}.mass2geo < payload.mass)
            continue;
        end
    end
    if(strcmp(payload.orbit_type,'polar') == 1)
        if(veh{i}.mass2polar < payload.mass)
            continue;
        end
    end
    if(strcmp(payload.orbit_type,'sunsync') == 1)
        if(veh{i}.mass2sunsync < payload.mass)
            continue;
        end
    end
    if(strcmp(payload.orbit_type,'ss') == 1)
```

```
        if(veh{i}.mass2ss < payload.mass)
            continue;
        end
    end
    %diameter constraint
    if(veh{i}.fairingdiameter < payload.diameter)
        continue;
    end
    %height constraint
    if(veh{i}.fairingheight < payload.height)
        continue;
    end
    %max_acceleration constraint
    if(payload.max_lat_accel ~= -1)
        if(veh{i}.max_lat_accel > payload.max_lat_accel)
            continue;
        end
    end
    %max_acceleration constraint
    if(payload.max_axial_accel ~= -1)
    if(veh{i}.max_axial_accel > payload.max_axial_accel)
        continue;
    end
end
    %vibration constraint
    if(payload.lat_freq ~= -1)
        if(veh{i}.min_lat_freq > payload.lat_freq)
            continue;
        end
    end
    if(payload.long_freq ~= -1)
        if(veh{i}.min_long_freq > payload.long_freq)
            continue;
        end
    end
    %acoustic constraint
    if(payload.acoustic ~= -1)
    if(veh{i}.acoustic > payload.acoustic)
        continue;
    end
end
    %temp constraint
    if(payload.max_aeroheating ~= -1)
    if(veh{i}.max_aeroheating > payload.max_aeroheating)
        continue;
    end
end
    %air_clean constraint
    if(payload.air_clean ~= -1)
        if(veh{i}.air_clean < payload.air_clean)
            continue;
        end
    end
    %orbit_accu
    if(payload.orbital_accu_alt ~= -1)
    if(veh{i}.orbital_accu_alt > payload.orbital_accu_alt)
        continue;
    end
end
    if(payload.orbital_accu_incl ~= -1)
    if(veh{i}.orbital_accu_incl > payload.orbital_accu_incl)
        continue;
    end
end

    %vehicle passes all constraints, add to list
    qualified_vehicles{m} = veh{i};
%     fprintf(['\n' qualified_vehicles{m}.class '\n'])
    m = m+1;
end

%calculate reliability
%calculate availiability for remaining vehicles
%calculate performance margin
if(exist('qualified_vehicles') == 0)
    error('  No Vehicles Qualify.');
end

for i = [1:length(qualified_vehicles)]
       if(qualified_vehicles{i}.total_flights == 0)
           qualified_vehicles{i}.reliability = 0;
       else
           qualified_vehicles{i}.reliability = qualified_vehicles{i}.success_flight/qualified_vehicles{i}.total_flights;
```

```
       end
       qualified_vehicles{i}.availiability = 1 - (payload.launchfreq*(1-
qualified_vehicles{i}.reliability)*qualified_vehicles{i}.stddwntime/(1 - 1/qualified_vehicles{i}.surge));
    if(strcmp(payload.orbit_type,'leo') == 1)
        qualified_vehicles{i}.pmargin = qualified_vehicles{i}.mass2leo - payload.mass;
    end
    if(strcmp(payload.orbit_type,'gto') == 1)
        qualified_vehicles{i}.pmargin = qualified_vehicles{i}.mass2gto - payload.mass;
    end
    if(strcmp(payload.orbit_type,'geo') == 1)
        qualified_vehicles{i}.pmargin = qualified_vehicles{i}.mass2geo - payload.mass;
    end
    if(strcmp(payload.orbit_type, 'polar') == 1)
        qualified_vehicles{i}.pmargin = qualified_vehicles{i}.mass2polar - payload.mass;
    end
    if(strcmp(payload.orbit_type, 'sunsync') == 1)
        qualified_vehicles{i}.pmargin = qualified_vehicles{i}.mass2sunsync - payload.mass;
    end
    if(strcmp(payload.orbit_type, 'ss') == 1)
        qualified_vehicles{i}.pmargin = qualified_vehicles{i}.mass2ss - payload.mass;
    end
end

% now optimize based on options
m = 1;
if(payload.absolute_cost == 1)
    fprintf('\n## Absolute Cost Optimization ##\n')
    while(~isempty(qualified_vehicles))
        min = qualified_vehicles{1}.cost+((1 - qualified_vehicles{1}.reliability)*qualified_vehicles{1}.cost);
        bestveh{m} = qualified_vehicles{1};
        bestveh{m}.abs_cost = min;

        best = 1;
        for i = [1:length(qualified_vehicles)]
            if(qualified_vehicles{i}.cost + ((1 - qualified_vehicles{i}.reliability)*qualified_vehicles{i}.cost) < min)
                bestveh{m} = qualified_vehicles{i};
                bestveh{m}.abs_cost = (qualified_vehicles{i}.cost + ((1 -
qualified_vehicles{i}.reliability)*qualified_vehicles{i}.cost));
                min = qualified_vehicles{1}.cost+((1 - qualified_vehicles{1}.reliability)*qualified_vehicles{1}.cost);
                best = i;
            end
        end
        %delete best entry from qualified vehicles
        fprintf(['\n   ' qualified_vehicles{best}.class ' | ' num2str(qualified_vehicles{best}.cost+((1 -
qualified_vehicles{best}.reliability)*qualified_vehicles{best}.cost)) '\n'])
        qualified_vehicles(best) = [];
        m = m+1;
    end
end
if(payload.cost == 1)
    fprintf('\n## Cost Optimization ##\n')
    while(~isempty(qualified_vehicles))
        min = qualified_vehicles{1}.cost;
        bestveh{m} = qualified_vehicles{1};
        best = 1;
        for i = [1:length(qualified_vehicles)]
            if(qualified_vehicles{i}.cost < min)
                bestveh{m} = qualified_vehicles{i};
                min = qualified_vehicles{i}.cost;
                best = i;
            end
        end
        %delete best entry from qualified vehicles
        fprintf(['\n   ' qualified_vehicles{best}.class ' | ' num2str(qualified_vehicles{best}.cost) '\n'])
        qualified_vehicles(best) = [];
        m = m+1;
    end
end
if(payload.risk == 1)
    fprintf('\n## Risk Optimization ##\n')
    while(~isempty(qualified_vehicles))
        min = (1 - qualified_vehicles{1}.reliability)*qualified_vehicles{1}.cost;
        bestveh{m} = qualified_vehicles{1};
        best = 1;
        for i = [1:length(qualified_vehicles)]
            if((1 - qualified_vehicles{i}.reliability)*qualified_vehicles{i}.cost < min)
                bestveh{m} = qualified_vehicles{i};
                min = (1 - qualified_vehicles{i}.reliability)*qualified_vehicles{i}.cost;
                best = i;
            end
        end
        %delete best entry from qualified vehicles
```

```
        fprintf(['\n   ' qualified_vehicles{best}.class ' | ' num2str((1 -
qualified_vehicles{best}.reliability)*qualified_vehicles{best}.cost) '\n'])
        qualified_vehicles(best) = [];
        m = m+1;
    end
end
if(payload.availiability == 1)
    fprintf('\n## Availability Optimization ##\n')
    while(~isempty(qualified_vehicles))
        max = qualified_vehicles{1}.availiability;
        bestveh{m} = qualified_vehicles{1};
        best = 1;
        for i = [1:length(qualified_vehicles)]
            if(qualified_vehicles{i}.availiability > max)
                bestveh{m} = qualified_vehicles{i};
                max = qualified_vehicles{i}.availiability;
                best = i;
            end
        end
        %delete best entry from qualified vehicles
        fprintf(['\n   ' qualified_vehicles{best}.class ' | ' num2str(qualified_vehicles{best}.availiability) '\n'])

        qualified_vehicles(best) = [];
        m = m+1;
    end
end
if(payload.pmargin == 1)
    fprintf('\n## Performance Margin Optimization ##\n')
    while(~isempty(qualified_vehicles))
        max = qualified_vehicles{1}.pmargin;
        bestveh{m} = qualified_vehicles{1};
        best = 1;
        for i = [1:length(qualified_vehicles)]
            if(qualified_vehicles{i}.pmargin > max)
                bestveh{m} = qualified_vehicles{i};
                max = qualified_vehicles{i}.pmargin;
                best = i;
            end
        end
        %delete best entry from qualified vehicles
        fprintf(['\n   ' qualified_vehicles{best}.class ' | ' num2str(qualified_vehicles{best}.pmargin) '\n'])
        qualified_vehicles(best) = [];
        m = m+1;
    end
end
```