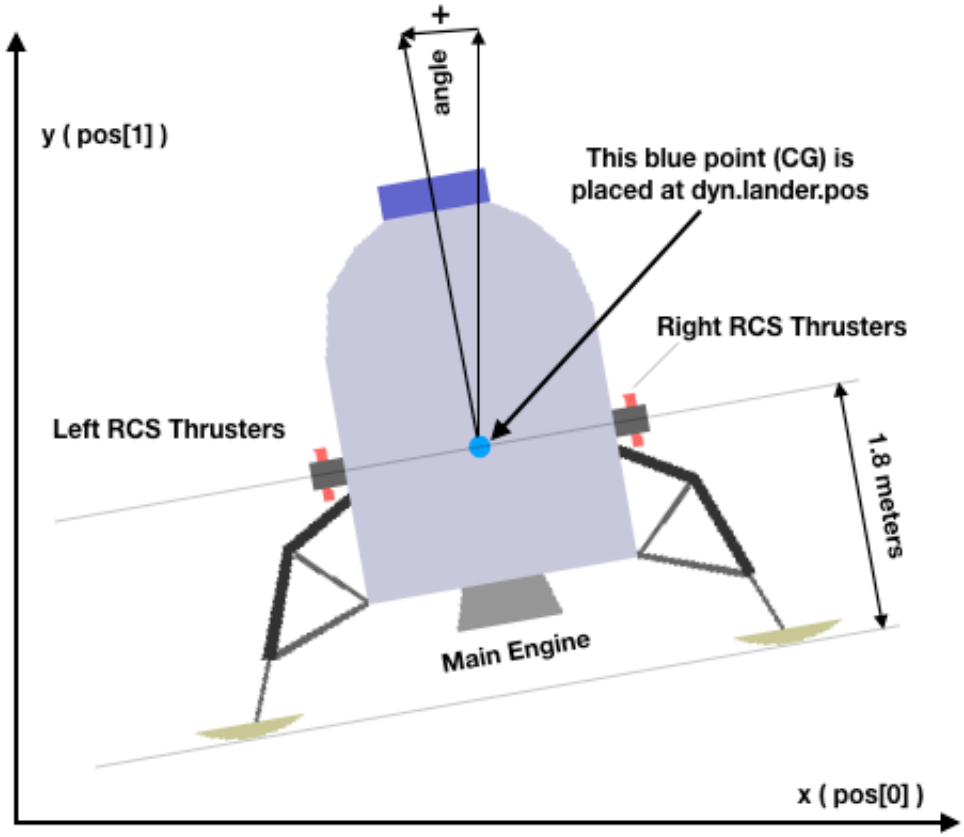
Lander Simulation Assignment

Create a lander / hopper simulation, with graphics under the forces of thrust, gravity, and under torque of RCS thruster.

1. the parameters and equations shown below.
2. the supplied variable server graphics client.

- Step 1: Lander should be able to lift off and land on the ground, straight up and down using manual thrust controls.
- Step 2: Add RCS (roll-left and roll right) manual control.
- Step 3: Modify the sim to have a fixed amount of fuel.
- Step 4: Add something that you think would be cool. Perhaps altitude-hold, which maintains a particular altitude when AUTO_1 in on.



Suggested Lander Data Members

```
        // Navigation State
        double pos[2] ;    /* (m)     xy-position */
        double vel[2] ;    /* (m/s)   xy-velocity */
        double acc[2] ;    /* (m/s2)  xy-acceleration  */
        double angle ;     /* (rad)   CCW = positive, CW = negative */
        double angleDot;   /* (rad/s) */
        double angleDDot;  /* (rad/s2) */

        // Vehicle Controls State
        int throttle;             /* Range 0..100 (percent) */
        int rcs_command;          /* 1 = CCW, 0 = no torque, -1 = CW */

        // Vehicle Attributes
        double thrust_max;        /* (N) Maxthrust the main engine can produce. */
        double rcs_torque;        /* Magnitude of RCS torque. */
        double mass;              /* (kg) */
        double moment_of_inertia; /* (kg/m2) */
        double g;                 /* (m/s2) - acceleration of gravity. */

        // Controls from the client
        int manual_throttle_change_command; /* (percent) update to throttle. */
        int manual_rcs_command;             /* 1 = CCW, 0 = no torque, -1 = CW */
        int Auto_1;
        int Auto_2;
```

Also suggest you call sim object "dyn", as in cannon ball sim.

Vehicle Attributes

Name	Units	Value
thrust_max	N	15000
rcs_torque		50
mass	kg	2000
moment_of_inertia	kg/m2	2000
g	m/s2	1.62

Initial State

Name	Units	Value
pos[2]	(m)	0.0, 1.8
vel[2]	(m/s)	0.0, 0.0
angle	rad	0.0
angleDot	rad	0.0

NOTE: **pos** is the position of the center of the lander. When the lander is sitting on the ground, pos[1] is 1.8 meters above the ground.

State Derivatives

[Eq#] **thrust** = (throttle / 100) * thrust_max

[Eq#] **acc[0]** = (thrust * -sin(angle) * thrust_max

[Eq#] **acc[1]** = (thrust * cos(angle) * thrust_max

[Eq#] **angleDDot** = rcs_command * rcs_torque / moment_of_inertia

State Integration

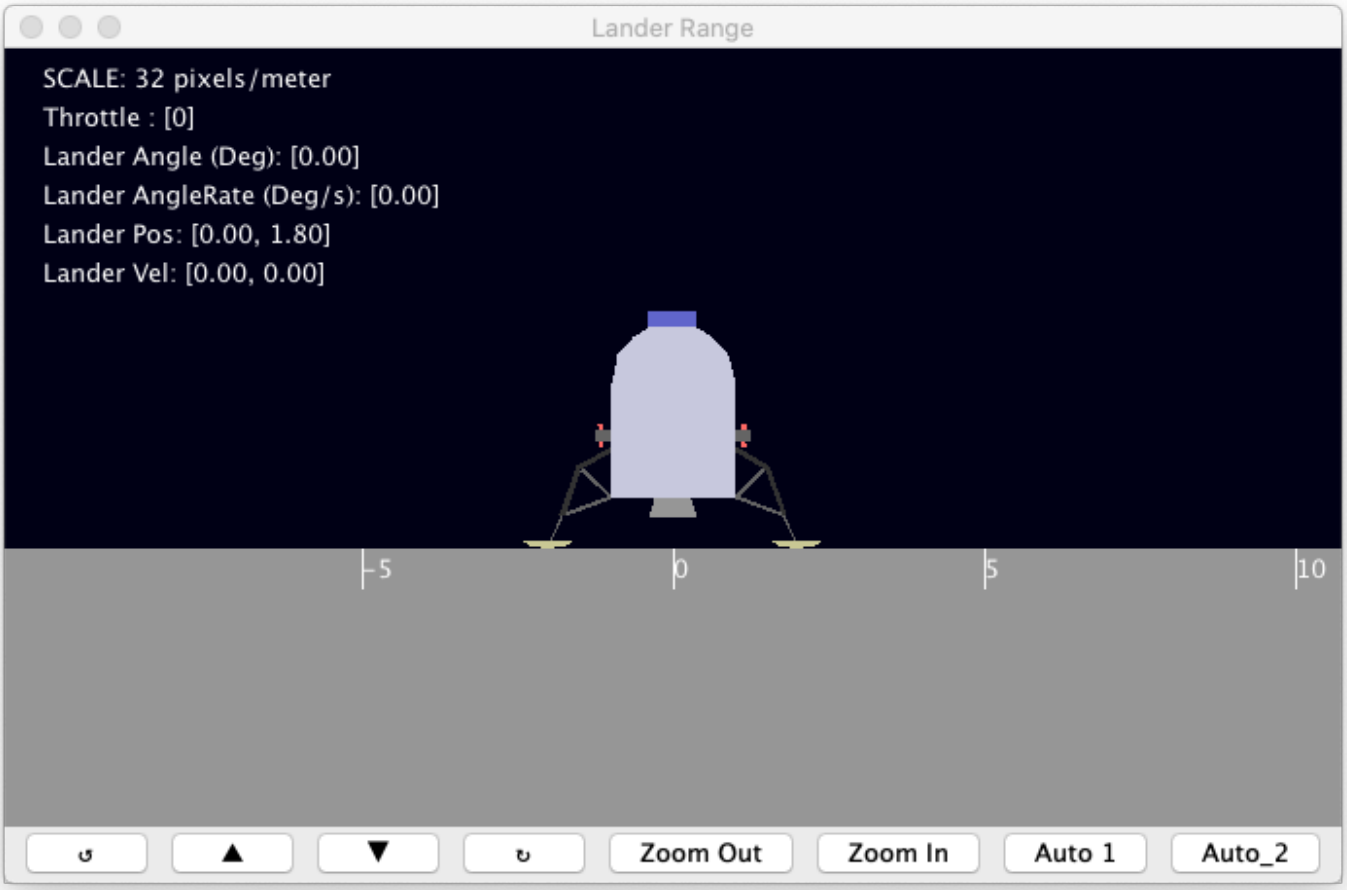
Name	Units	Value
pos[2]	(m)	Calculated by numerical integration of vel.
vel[2]	(m/s)	Calculated by numerical integration of acc.
angle	rad	Calculated by numerical integration of angleDot.
angleDot	rad	Calculated by numerical integration of angleDDot.

Controls-State Update

Name	Units	Value
throttle	--	throttle = manual_throttle_change_command

throttle = manual_throttle_change_command

Lander Variable Server Client



Button	Description
↺	Set `dyn.lander.manual_rcs_command = 1;`
▲	Set `dyn.lander.manual\_throttle\_change\_command = 1;`
▼	Set `dyn.lander.manual\_throttle\_change\_command = -1;`
↻	Set `dyn.lander.manual_rcs_command = -1;`
Zoom Out	Zoom out the lander graphics.
Zoom In	Zoom in the lander graphics.
Auto 1	Toggle `dyn.lander.Auto_1` between 0 [off], and 1 [on]
Auto 2	Toggle `dyn.lander.Auto_2` between 0 [off], and 1 [on]

Controls from Client

Name	Units	Value
manual_throttle_change_command	--	0
manual_rcs_command	--	0
Auto_1	--	0
Auto_2	--	0

Variable Values to Send from SIM to the Graphics Client.

```
        evd.out.writeBytes( "trick.var_pause() \n" +
                            "trick.var_add(\"dyn.lander.pos[0]\")\n" +
                            "trick.var_add(\"dyn.lander.pos[1]\")\n" +
                            "trick.var_add(\"dyn.lander.angle\")\n" +
                            "trick.var_add(\"dyn.lander.vel[0]\")\n" +
                            "trick.var_add(\"dyn.lander.vel[1]\")\n" +
                            "trick.var_add(\"dyn.lander.angleDot\")\n" +
                            "trick.var_add(\"dyn.lander.throttle\")\n" +
                            "trick.var_add(\"trick_sys.sched.mode\")\n" +
                            "trick.var_ascii() \n" +
                            String.format("trick.var_cycle(%.3f)\n", dt) +
                            "trick.var_unpause() \n" );
        evd.out.flush();
```

Set SIM Variable from the Graphics Client.

```
        throttle_delta = rangeView.getThrottleDelta();
        evd.out.writeBytes( String.format(
        "dyn.lander.manual_throttle_change_command = %d ;\n", throttle_delta ));
        rangeView.throttleZeroDelta();

        rcs_state = rangeView.get_rcs_state();
        evd.out.writeBytes( String.format(
        "dyn.lander.manual_rcs_command = %d ;\n", rcs_state ));
        rangeView.rcs_zero();

        if (rangeView.getAuto_1()) Auto_1 = 1; else Auto_1 = 0;
        evd.out.writeBytes( String.format(
        "dyn.lander.Auto_1 = %d ;\n", Auto_1 ));

        if (rangeView.getAuto_2()) Auto_2 = 1; else Auto_2 = 0;
        evd.out.writeBytes( String.format(
        "dyn.lander.Auto_2 = %d ;\n", Auto_2 ));
```

`"post-integration"` Jobs

"post-integration" jobs, as their name indciates, run immediately after "integration" jobs. In these jobs you can change the state as needed. Maybe you want to limit the position or velocity or angle or ...

Hint:

```
    if (pos[1] < 1.8) {
        // How might you want to set the state when (pos[1] < 1.8)?
    }
```

`"scheduled"` Jobs

```
    if (Auto_1 == 1) {
        // Do some automatic control stuff ... (e.g. altitude hold).
    }

    if (Auto_2 == 1) {
        // Do some automatic control stuff ... (up to you also).
    }
```