NASA Artemis Student Challenge Opportunities

Software Challenge Informational Session #3 for Competition Teams

**Student Team**
- UC San Diego
- University of Southern California
- Berkeley

**Faculty Team**
- John Kosmatka (PI)
  University of California, San Diego
- David Barnhart (Co-PI)
  University of Southern California
- Dan Zevin
  University of California Berkeley

May 12th, 2021
Reminder:
LEAPFROG is 1 of 3 Theme-3 2020 Awards

**THEME 3 AWARDS – PILOT ARTEMIS STUDENT CHALLENGES**

- Lunar/Martian Lander skills competition using existing technology to execute the competition in Earth’s gravity and atmosphere
- Develops Artemis-relevant systems engineering and integration skills, and requires innovations to perform well against the other teams
- Requires the development and evolution of critical thinking and hands-on skill sets to enable humankind’s next great steps off of Earth
Artemis LEAPFROG Project Goals

- Update existing USC LEAPFROG lander prototype
- Develop competitions around the Lunar Lander prototype
- Run Competitions in Summer of 2021!

Run a national competition in 2021!
<table>
<thead>
<tr>
<th>Activities</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Signups for all potential Teams</td>
<td>3/5/21</td>
<td>5/7/21</td>
</tr>
<tr>
<td>Run PX4 and Simulation Webinars</td>
<td>3/28/21</td>
<td>5/12/21</td>
</tr>
<tr>
<td>Finalize Teams, Notify Acceptance</td>
<td>5/12/21</td>
<td>5/17/21</td>
</tr>
<tr>
<td>Seminar #1 S/W Tools</td>
<td></td>
<td>4/7/2021</td>
</tr>
<tr>
<td>Seminar #2 Running the Sim</td>
<td></td>
<td>4/30/2021</td>
</tr>
<tr>
<td>Seminar #3 Modifying Sim</td>
<td></td>
<td>5/12/2021</td>
</tr>
<tr>
<td>Seminar #4 Competition Metrics</td>
<td></td>
<td>5/21/2021</td>
</tr>
<tr>
<td>Run Software Challenge (4 Weeks)</td>
<td>5/24/21</td>
<td>6/18/21</td>
</tr>
<tr>
<td>Finalize and Contact Winners</td>
<td>6/21/21</td>
<td>6/30/21</td>
</tr>
<tr>
<td>Ship out LEAPFROG Vehicle Kits</td>
<td>7/1/21</td>
<td>7/15/21</td>
</tr>
<tr>
<td>Run Vehicle Building Bootcamps &amp; Run</td>
<td>7/28/21</td>
<td>8/11/21</td>
</tr>
<tr>
<td>Exhibition Flights</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Submit Final Report to NASA</td>
<td>8/5/21</td>
<td>8/31/21</td>
</tr>
</tbody>
</table>
Update on Initial Team’s (as of 4/30/2021)

Current Teams identified in our Registration Site:

<table>
<thead>
<tr>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idaho State University</td>
</tr>
<tr>
<td>Purdue University</td>
</tr>
<tr>
<td>New Mexico Tech</td>
</tr>
<tr>
<td>City College of San Francisco</td>
</tr>
<tr>
<td>New Mexico Highlands University</td>
</tr>
<tr>
<td>California State Polytechnic University, Pomona</td>
</tr>
<tr>
<td>University of Illinois Urbana-Champaign</td>
</tr>
<tr>
<td>UC Berkeley</td>
</tr>
<tr>
<td>University of West Florida</td>
</tr>
<tr>
<td>Arizona State University</td>
</tr>
<tr>
<td>University of Texas at Austin</td>
</tr>
<tr>
<td>Texas A&amp;M University</td>
</tr>
<tr>
<td>New Mexico State University</td>
</tr>
</tbody>
</table>

Notes:
- Only US universities or institutions can sign up for possible Lander Kits
- International participation is possible as part of a USC team
- Every Team MUST have a Faculty Mentor signed up
Software Competition Challenge and Exhibition Flights

~ March to June 2021

**Virtual Software Challenge**

**Crater Landing**

**Goal:** Locate & land in center of designated crater

**Obstacle Avoidance**

**Goal:** Identify & maneuver past potential hazards

~ June to August 2021

**Kit Boot Camp and Flight Exhibition**

**Goal:** Enable teams from S/W challenge to build full kit (without Engine), and run Exhibition Flights with Winning S/W

• Virtual S/W Challenge run remotely
• Skills enable transfer between Simulation and Flight
  • Waypoint navigation, control methodology, obstacle avoidance
• Two Challenges earn Points, highest point earner wins
• Top 6 teams will advance to Hardware Kit Build and Top 3 for Flight Exhibition
• Possible that more kits may be available to competition teams (US only)

• Teams from S/W Challenge (first 6 at minimum) will receive full hardware “Kit”
• Boot Camp will be run during summer to help all teams “Build” their Lunar Lander Vehicle
• Exhibition Flights will be run in CA for the Top three Teams in the S/W challenge
Software Challenge uses PixHawk Simulation Software

- Basic simulation with plant model of LEAPFROG components created
- Generic Earth centric physics model applied, uses Gazebo for visualization
- PixHawk controller algorithms integrated with open source tools into simulation
- Scenario provided to mimic dimensions and geography of competition RC Field

• The bridge between the Software and Flight Challenge is the PIXHAWK CONTROLLER
Initial Plan for Teams to share Simulation Code
(Presented in Seminar #4 and will be put on the Website)
Where to get more Information?

- "LEAPFROG.ISI.EDU"

- Competition Information will be provided
- Vehicle build updates
- Link to the Registration
- Links to NASA Artemis STEM Project Information
- Information about the LEAPFROG Team members
Whom to Contact

• Fill out registration on Website (leapfrog.isi.edu)

• Information E-mail: leapfrog@usc.edu

• Technical and Competition Contact:
  • Prof David Barnhart, Barnhart@isi.edu

• Programmatic Contact:
  • Prof John Kosmatka, jkosmatka@ucsd.edu
DETAILS FOR MODIFYING AND EXTENDING PLUGINS FOR THRUST VECTOR CONTROL, ACS AND NAVIGATION ALGORITHMS
LEAPFROG Webinar 3: Extending and Modifying

May 12, 2021
Today’s Overview

1. Implementing TVC and ACS Plugin
2. Adding New Plugin
3. Recap/References
Implementing TVC and ACS Plugin

Overall steps for making a TVC or ACS plugin has the following steps:

1. Obtain link or joint from Gazebo model to control
2. Create a controller that will output the correct value to the link or joint
3. Apply output of controller to actual link or joint
TVC Plugin Example

1. Obtain joint from Gazebo model to control

```cpp
55   // Get linear actuator joints (from lander.sdf)
56   actuator_1_joint_name = "actuator_1_outer__actuator_1_inner__prismatic";
57   actuator_2_joint_name = "actuator_2_outer__actuator_2_inner__prismatic";
58
59   actuator_1_joint = model_\rightarrow GetJoint(actuator_1_joint_name);
60   actuator_2_joint = model_\rightarrow GetJoint(actuator_2_joint_name);
```

Link to lines in TVC plugin [here](#)
TVC Plugin Example cont.

2. Create a controller that will output the correct value to the link or joint
   a. We have provided the current status of the linear actuator
   b. Design and implementation of controller needs to be done
   c. In example, output of controller should could go into
      \_actuator\_status\_1 and \_actuator\_status\_2

```cpp
void TVCCcontrollerPlugin::handle_control() {
  // Get current linear actuator joint positions
  _actuator\_current\_1 = actuator\_1\_joint\_->Position(0);
  _actuator\_current\_2 = actuator\_2\_joint\_->Position(0);

  // Get actual forces for linear actuators
  _actuator\_status\_1 = -1;
  _actuator\_status\_2 = -1;

  // Save velocity of linear actuators
  _actuator\_velocity\_1 = actuator\_1\_joint\_->GetVelocity(0);
  _actuator\_velocity\_2 = actuator\_2\_joint\_->GetVelocity(0);

  // Apply forces to linear actuators
  actuator\_1\_joint\_->SetForce(0, _actuator\_status\_1);
  actuator\_2\_joint\_->SetForce(0, _actuator\_status\_2);
}
```

Link to lines in TVC plugin [here](#)
TVC Plugin Example cont.

3. Apply output of controller to actual joint
   a. This has already been done and is unnecessary to change.

```cpp
void TVCControllerPlugin::handle_control()
{
    // Get current linear actuator joint positions
    _actuator_current_1 = actuator_1_joint->Position(0);
    _actuator_current_2 = actuator_2_joint->Position(0);

    // Get actual forces for linear actuators
    _actuator_status_1 = -1;
    _actuator_status_2 = -1;

    // Save velocity of linear actuators
    _actuator_velocity_1 = actuator_1_joint->GetVelocity(0);
    _actuator_velocity_2 = actuator_2_joint->GetVelocity(0);

    // Apply forces to linear actuators
    actuator_1_joint->SetForce(0, _actuator_status_1);
    actuator_2_joint->SetForce(0, _actuator_status_2);
}
```

Link to lines in TVC plugin here
ACS Plugin Example

1. Obtain link from Gazebo model to control
   a. Gazebo docs for getting child link

```cpp
// Get thrusters from model
static actuator acs_po("lander::thruster_1");
static actuator acs_sb("lander::thruster_3");
static actuator acs_st("lander::thruster_2");
static actuator acs_bo("lander::thruster_4");
acs_po.link = _model->GetChildLink(acs_po.path);
acs_sb.link = _model->GetChildLink(acs_sb.path);
acs_st.link = _model->GetChildLink(acs_st.path);
acs_bo.link = _model->GetChildLink(acs_bo.path);

const ignition::math::Vector3<double> &force = {0, 0, 0};

// Apply force to thrusters
acs_po.link->AddLinkForce(force);
acs_sb.link->AddLinkForce(force);
acs_st.link->AddLinkForce(force);
acs_bo.link->AddLinkForce(force);
```

Link to lines in ACS plugin here
ACS Plugin Example *cont.*

2. Create a controller that will output the correct value to the link or joint
   a. Example is a different implementation of applying a force to a link then the TVC plugin
   b. Gazebo docs on links [here](#)

```cpp
128 // Get thrusters from model
129 static actuator acs_po("lander::thruster_1");
130 static actuator acs_sb("lander::thruster_3");
131 static actuator acs_st("lander::thruster_2");
132 static actuator acs_bo("lander::thruster_4");
133
134 acs_po.link = _model->GetChildLink(acs_po.path);
135 acs_sb.link = _model->GetChildLink(acs_sb.path);
136 acs_st.link = _model->GetChildLink(acs_st.path);
137 acs_bo.link = _model->GetChildLink(acs_bo.path);
138
139 const ignition::math::Vector3<double> &force = {0, 0, 0};
140
141 // Apply force to thrusters
142 acs_po.link->AddLinkForce(force);
143 acs_sb.link->AddLinkForce(force);
144 acs_st.link->AddLinkForce(force);
145 acs_bo.link->AddLinkForce(force);
```

Link to lines in ACS plugin [here](#)
ACS Plugin Example cont.

3. Apply output of controller to actual link
   a. Determination of which forces to apply to the respective thrusters must be done.

```cpp
128 // Get thrusters from model
129 static actuator acs_po("lander::thruster_1");
130 static actuator acs_sb("lander::thruster_3");
131 static actuator acs_st("lander::thruster_2");
132 static actuator acs_bo("lander::thruster_4");
133
134 acs_po.link = _model->GetChildLink(acs_po.path);
135 acs_sb.link = _model->GetChildLink(acs_sb.path);
136 acs_st.link = _model->GetChildLink(acs_st.path);
137 acs_bo.link = _model->GetChildLink(acs_bo.path);
138
139 const ignition::math::Vector3<double> &force = {0, 0, 0};
140
141 // Apply force to thrusters
142 acs_po.link->AddLinkForce(force);
143 acs_sb.link->AddLinkForce(force);
144 acs_st.link->AddLinkForce(force);
145 acs_bo.link->AddLinkForce(force);
```

Link to lines in ACS plugin here
Adding New Plugin

1. Add plugin in **src** and **include** directories for source and header files respectively
   a. Generally follows normal C++ paradigm of separating declarations from actual implementation
Adding New Plugin \textit{cont.}

1. Add plugin in \textit{src} and \textit{include} directories for source and header files respectively
   a. Generally follows normal C++ paradigm of separating declarations from actual implementation
   b. Can follow hello_world.cc as a minimal example

```cpp
#include <gazebo/gazebo.hh>

namespace gazebo
{
    class WorldPluginTutorial : public WorldPlugin
    {
        public: WorldPluginTutorial() : WorldPlugin()
        {
            printf("Hello World!\n");
        }
        public: void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf)
        {
        }
    };
    GZ_REGISTER_WORLD_PLUGIN(WorldPluginTutorial)
}
```

\texttt{hello\_world.cc}
Adding New Plugin *cont.*

2. Add Plugin to CMakeLists.txt
   
   a. Register library: `source`

   ```cpp
   # custom libraries
   add_library(gazebo_acs_controller_plugin SHARED src/gazebo_acs_controller_plugin.cpp)
   add_library(gazebo_contact_plugin SHARED src/gazebo_contact_plugin.cc)
   add_library(gazebo_cg_plugin SHARED src/gazebo_cg_plugin.cpp)
   add_library(gazebo_custom_mavlink_interface SHARED src/gazebo_custom_mavlink_interface)
   add_library(gazebo_tvc_controller_plugin SHARED src/gazebo_tvc_controller_plugin.cpp)
   #
   ```
Adding New Plugin *cont.*

2. Add Plugin to CMakeLists.txt
   b. Add plugin to list of registered variables for cmake: `source`

```cpp
367    set(plUGINS
368          # custom plugins -----------------------------------------------
369    gazebo_acs_controller_plugin
370    gazebo_contact_plugin
371    gazebo_cg_plugin
372    gazebo_custom_mavlink_interface
373    gazebo_tvc_controller_plugin
374    # -----------------------------------------------
```
Adding New Plugin cont.

3. Add plugin to Gazebo model SDF
   a. This is necessary if the plugin created directly interacts with links or joints of the model
   b. Filename for plugin should correspond to name of library that was registered in the `CMakeLists.txt` file
   c. Example provided is the minimal example needed for TVC plugin: source

```xml
<plugin name='tvc_controller' filename='libgazebo_tvc_controller_plugin.so'>
  <robotNamespace/>
</plugin>
```
Adding New Plugin cont.

4. Verify that plugin is loaded in Gazebo
Recap

Implementing TVC and Controller Plugin
1. Obtain link or joint from Gazebo model to control
2. Create a controller that will output the correct value to the link or joint
3. Apply output of controller to actual link or joint

Creating New Plugin
1. Add plugin to src and include directories
2. Register plugin inside of CMakeLists.txt so that it will be compiled
3. Add plugin to Gazebo SDF for model so that it is loaded in simulation
4. Verify that model shows up in Gazebo
References

Implementing TVC and ACS Plugin

• TVC Controller Plugin Template
• ACS Controller Plugin Template
• Gazebo documentation for Link and Joint

Adding New Plugin

• Gazebo plugin example/documentation for hello world
• Simulation Repo
• CMakeLists.txt
• Gazebo model SDF