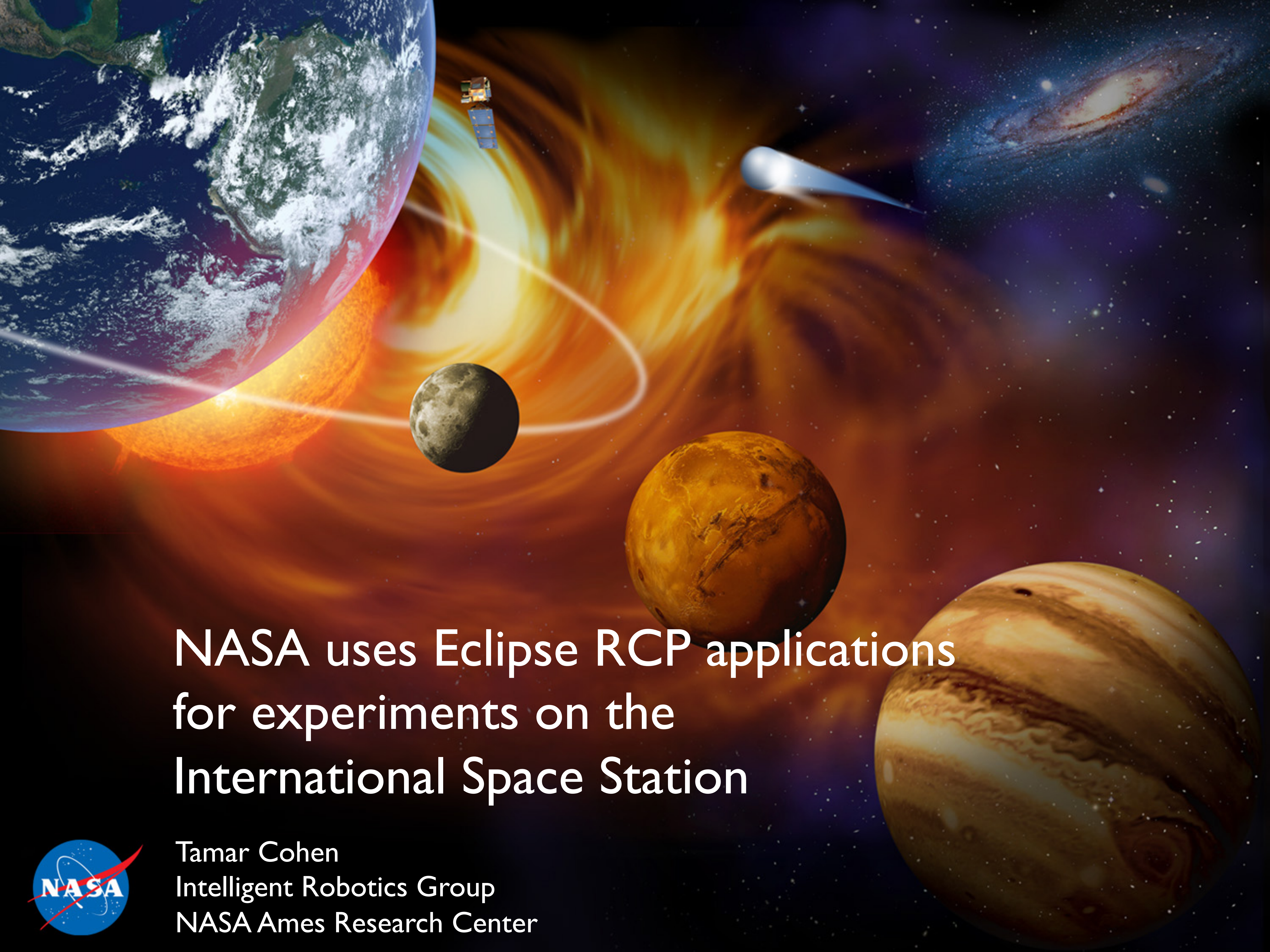


NASA uses Eclipse RCP applications for experiments on the International Space Station

Tamar Cohen
Intelligent Robotics Group
NASA Ames Research Center

In 2012 – 2013, the Intelligent Robotics Group from NASA Ames Research Center is conducting 2 experiments with the International Space Station (ISS)

Experiment 1: Simulate an internal inspection of a module of the ISS using the free-flying SPHERES robot with an Android Smartphone connected to it.

Experiment 2: Simulate deployment of a telescope by having an astronaut on the ISS control the K10 Rover at NASA Ames.

For both of these experiments, the astronauts will be using a custom "Workbench" RCP application. These are all based on Eclipse 3.7.2.

The ISS has a 450 page set of standards for software. This helps maintain consistency between various software control systems and helps astronauts with different native languages understand what various icons mean.

We also have to deal with unique usability issues, such as the fact that it is very difficult to click and point with a mouse when you are in zero g.

The only operating systems on the ISS computers with a GUI is currently Windows XP, so that is our target development platform. Internally we also use Linux and OSX, so we are doing cross-platform development.

Since we are developing multiple RCP applications we put common code into shared plugins. (This is one of the reasons we are still on Eclipse 3.7.2)

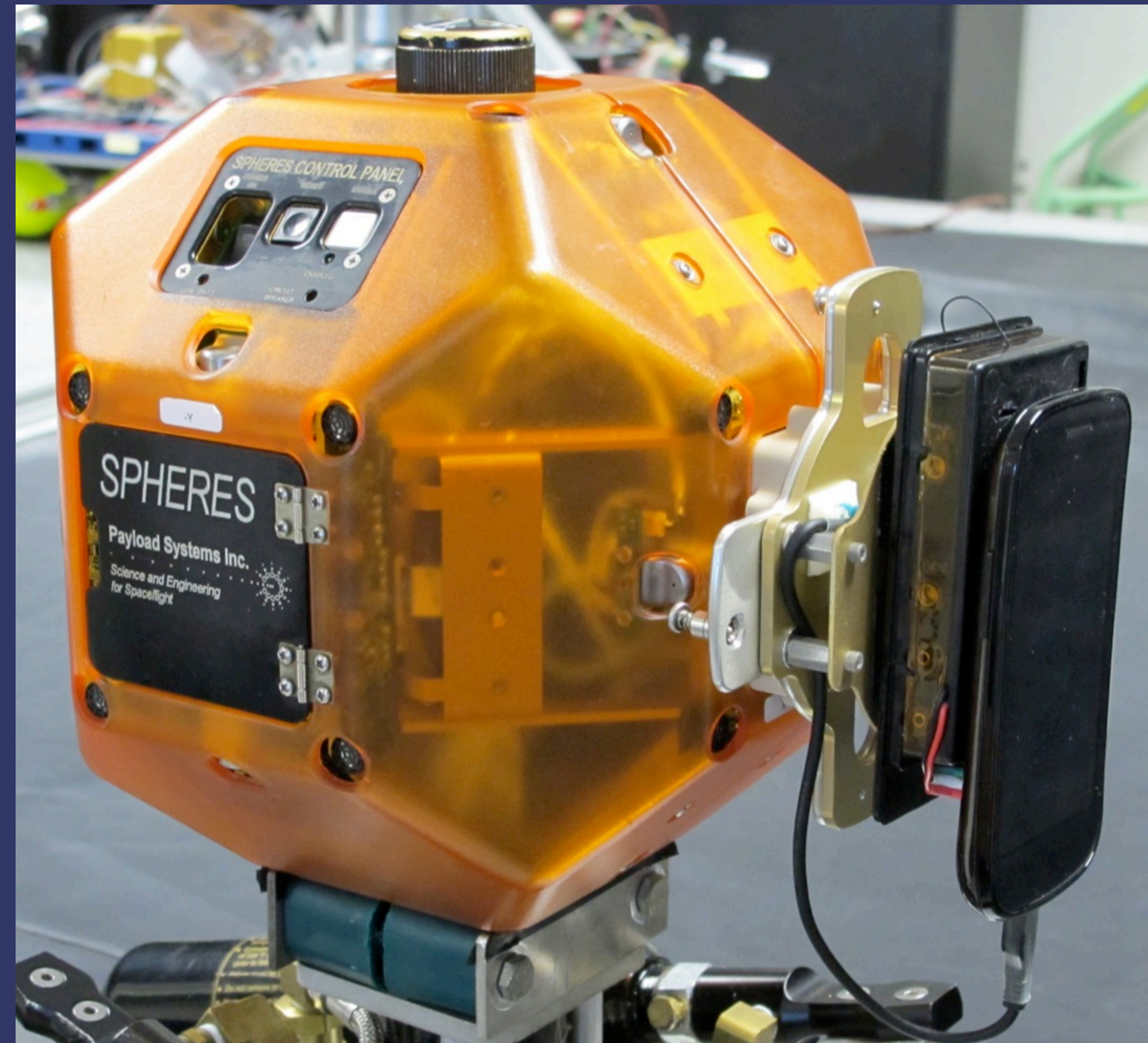
Experiment 1: SPHERES

SPHERES are free-flying satellite robots typically used for orbital experiments. SPHERES have been on the ISS since 2006, and were developed at MIT. (They do not include a Smartphone).

They use a cold-gas CO2 thruster system that is very similar to what is used on paintball guns. The entire system is powered using double-A batteries. A DSP microprocessor inside coordinates the mixing of the thrusters for the desired movement. The microprocessor also receives signals from five ultrasonic beacons, so the SPHERES can know where it is.

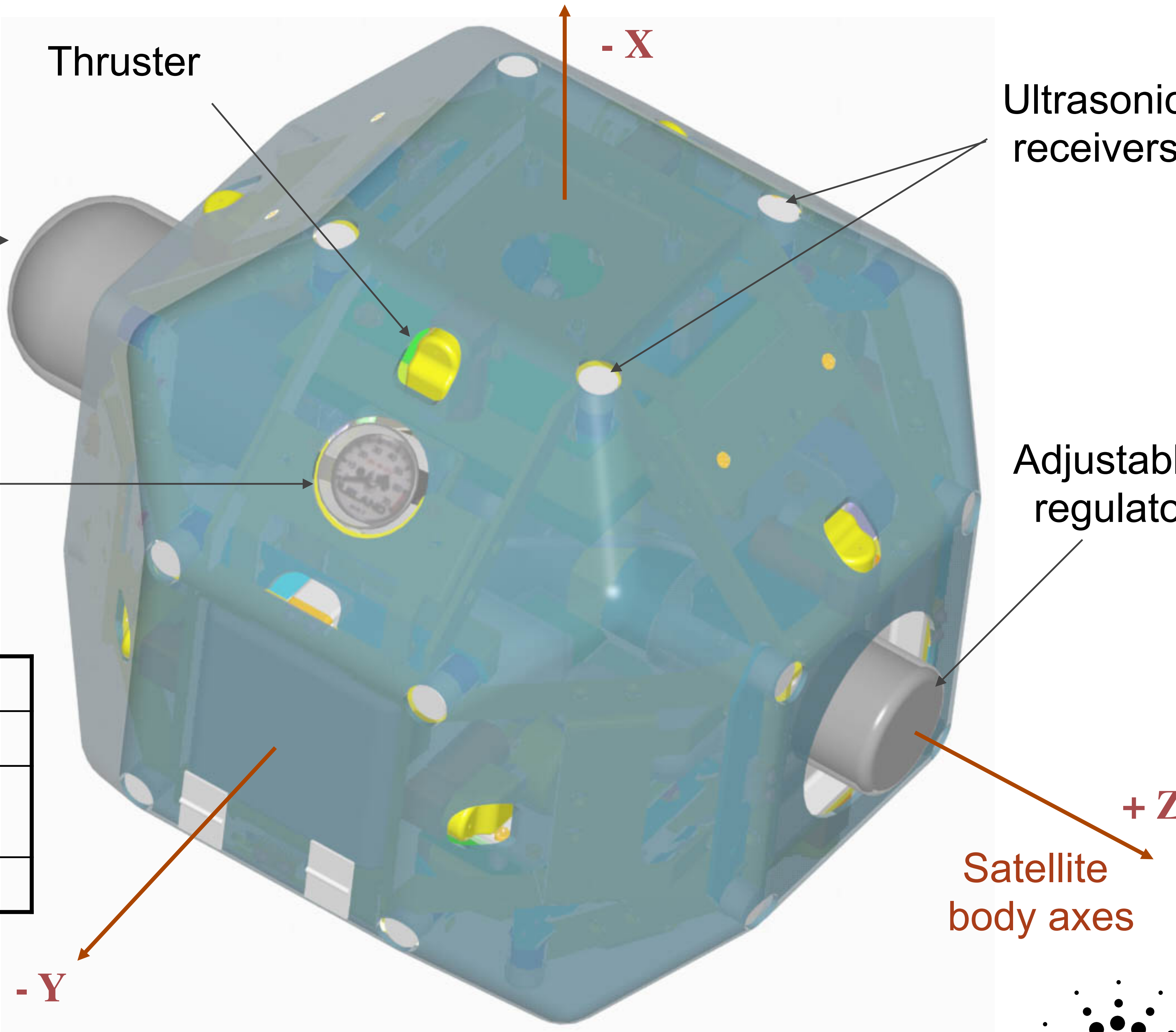
The SPHERES microprocessor is already fully taxed with normal SPHERES operations; we needed to add more processing power and a camera. We determined the most efficient way to do this was to adapt a Smartphone to work with the SPHERES.

We had to remove the battery and power it with AA batteries, put teflon tape over the screen, and remove the GPS chip, as well as put it through rigorous testing. Naturally we use velcro along with a custom USB cable to connect it to the SPHERES. We upmassed it on the last shuttle launch.



For the SPHERES experiments, we first controlled the SPHERES on the ISS from the SPHERES Smartphone Workbench RCP application running on Earth.
In the next iteration of this experiment, an astronaut on the ISS will control the SPHERES using the SPHERES Smartphone Workbench.

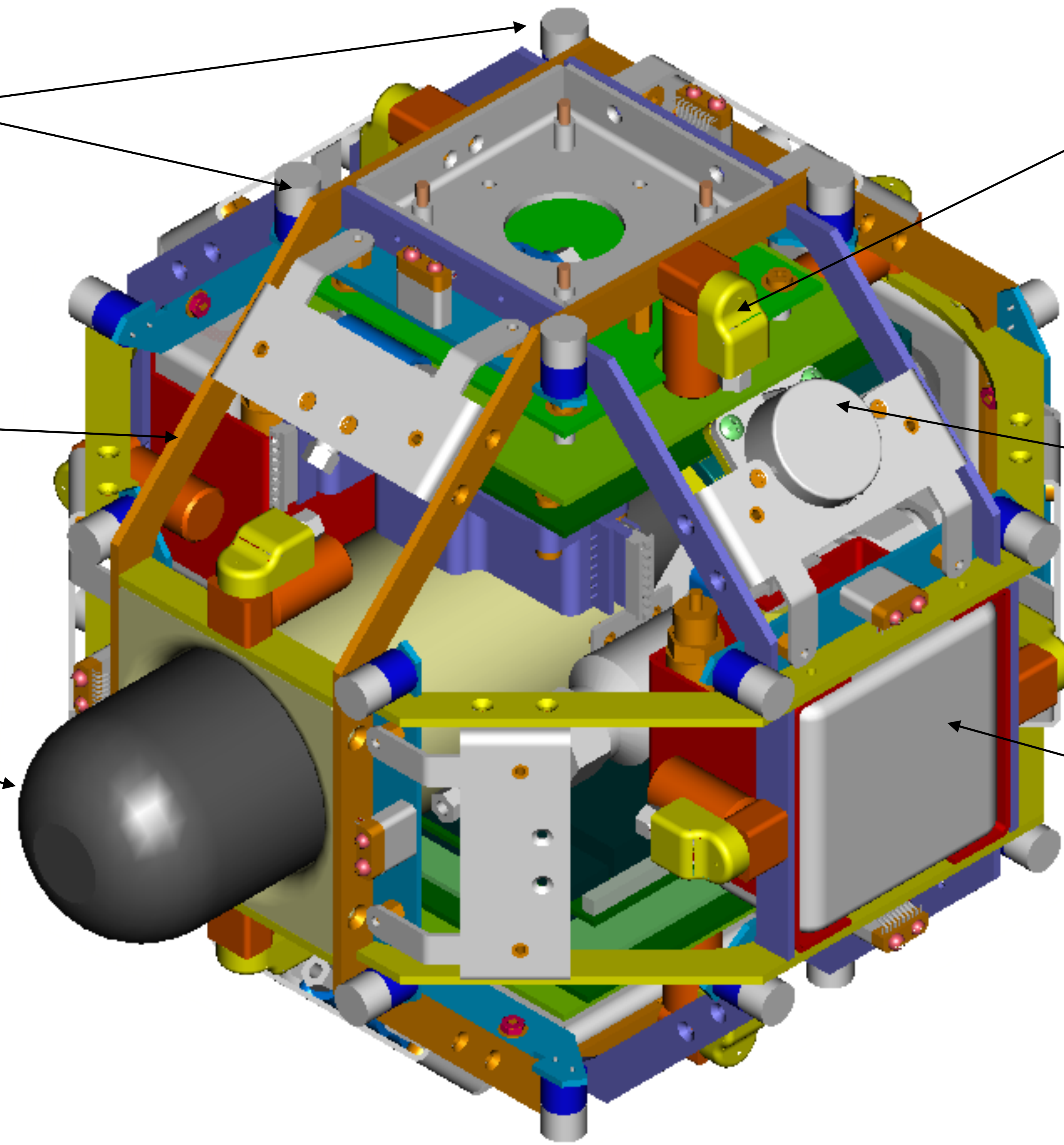
SPHERES Satellite



Diameter	8 in (0.2 m)
Mass	7.85 lb (3.56 kg)
Thrust (single thruster)	<1 oz (0.2 N)
CO_2 Capacity	6 oz (170g)

Structural Elements

- Satellite is fully functional without shell

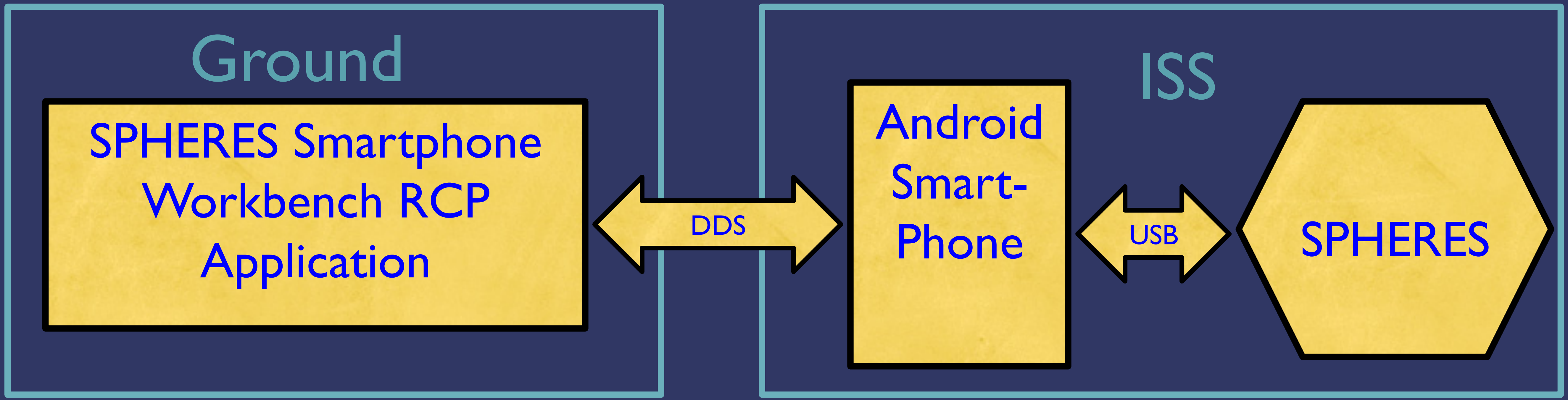


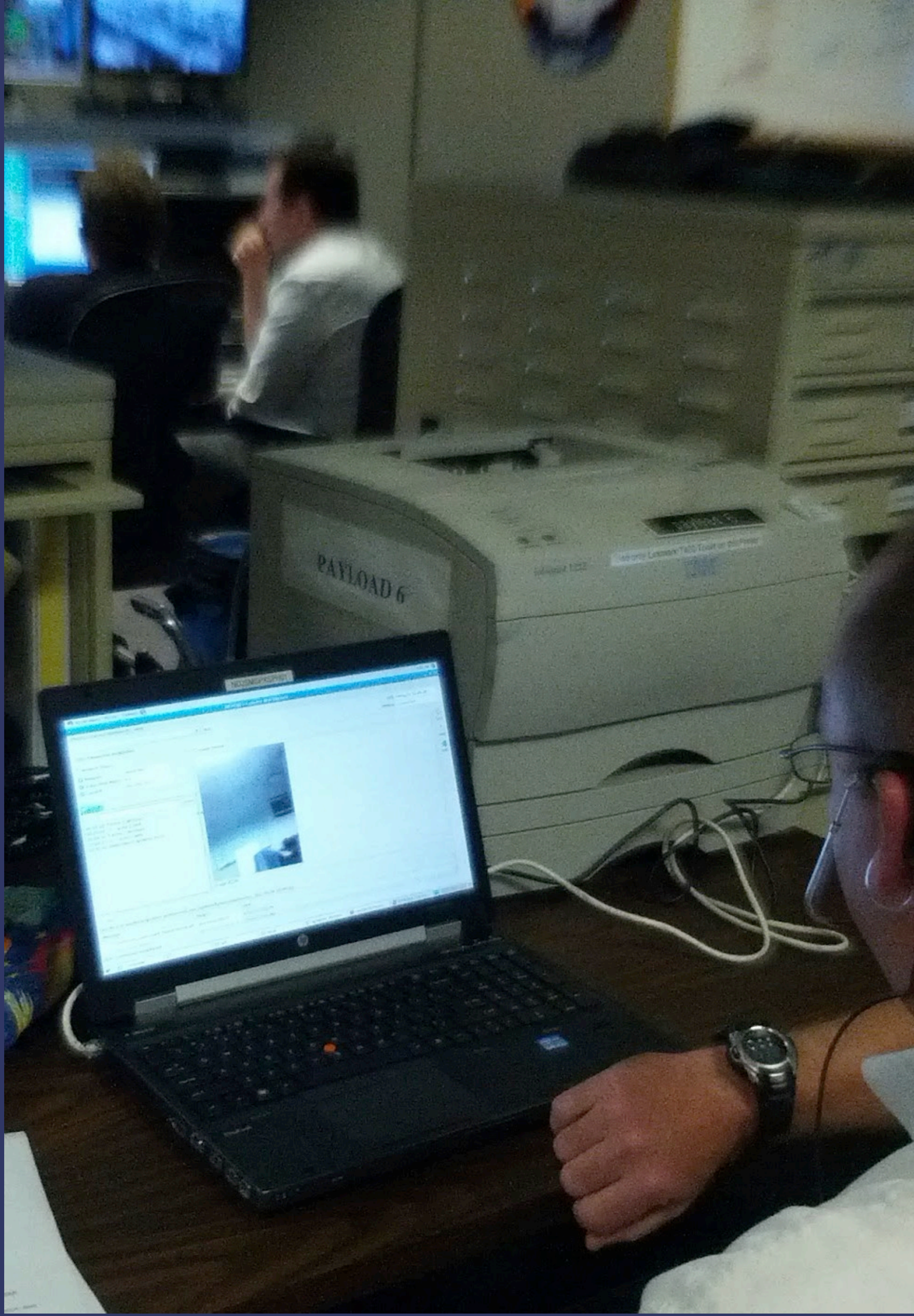
Slide Courtesy MIT and the Space Systems Laboratory

When we are commanding and monitoring robots over this long of a distance, we have a time delay between when commands are sent and when they are received; we have to design our software to account for this. We also have to support loss of signal (LOS) times, when the ISS is unable to communicate with Earth.

We use DDS, a data distribution system, to reliably send data. On computers, we use RTI's implementation of DDS, with our own standards, called RAPID, running on top of that. On the Android we licensed CoreDX DDS libraries.

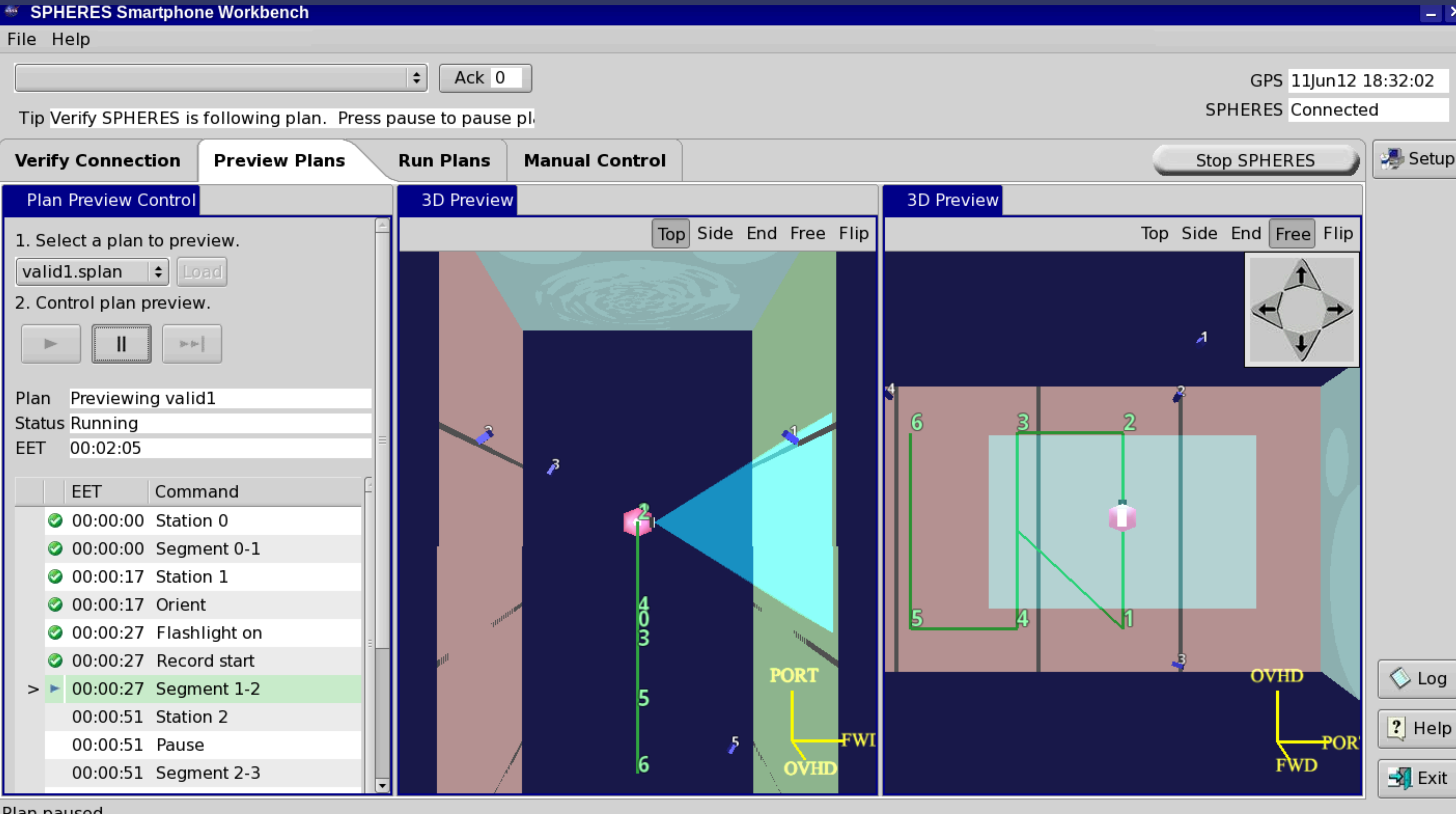
The SPHERES Smartphone Workbenc (RCP application) talks to the Android Smartphone, which communicates via USB cable to send commands to the SPHERES, and report the state back to the SPHERES Smartphone Workbench.



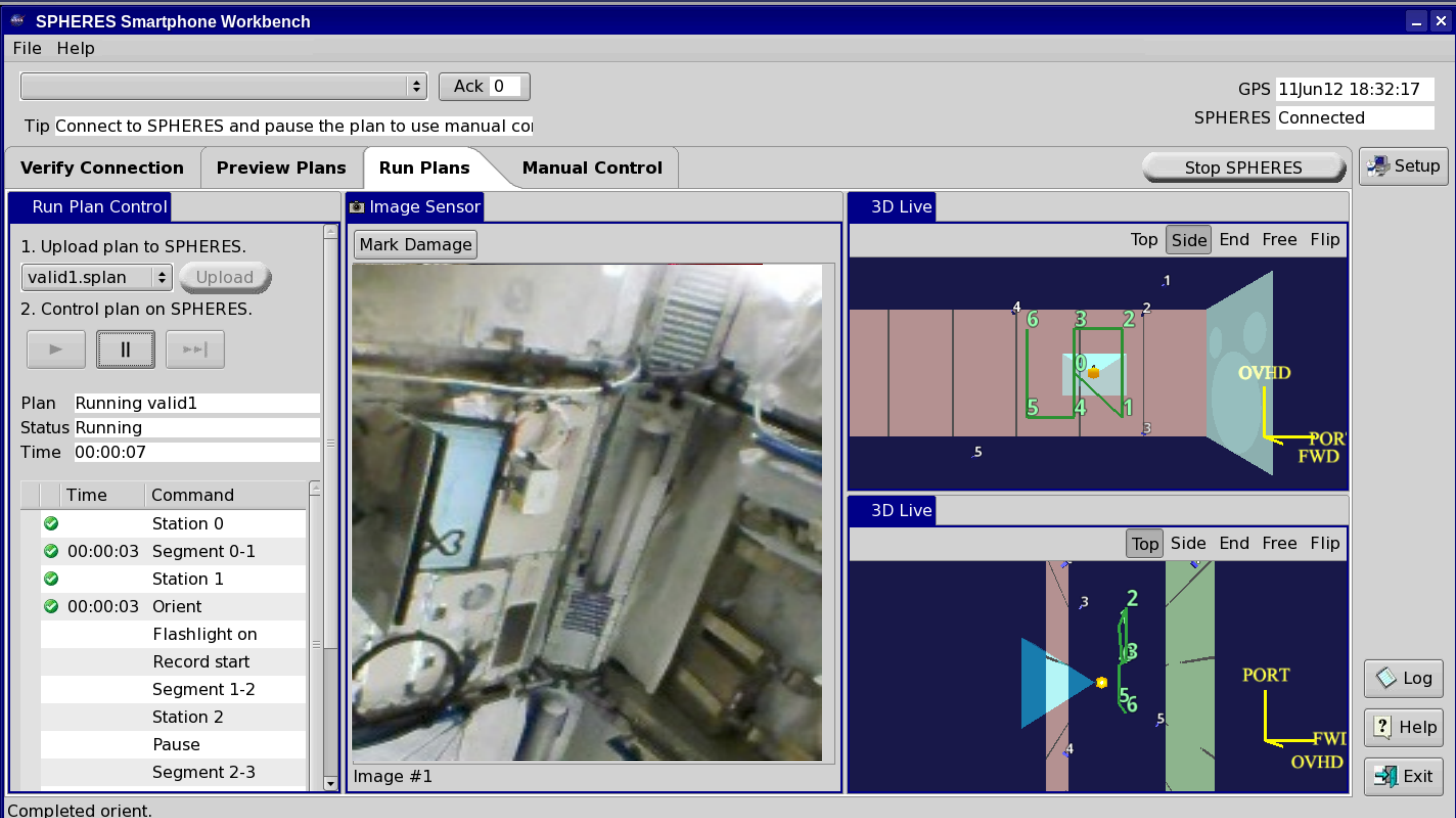

PAYLOAD 6

Screenshot of the SPHERES Smartphone Workbench

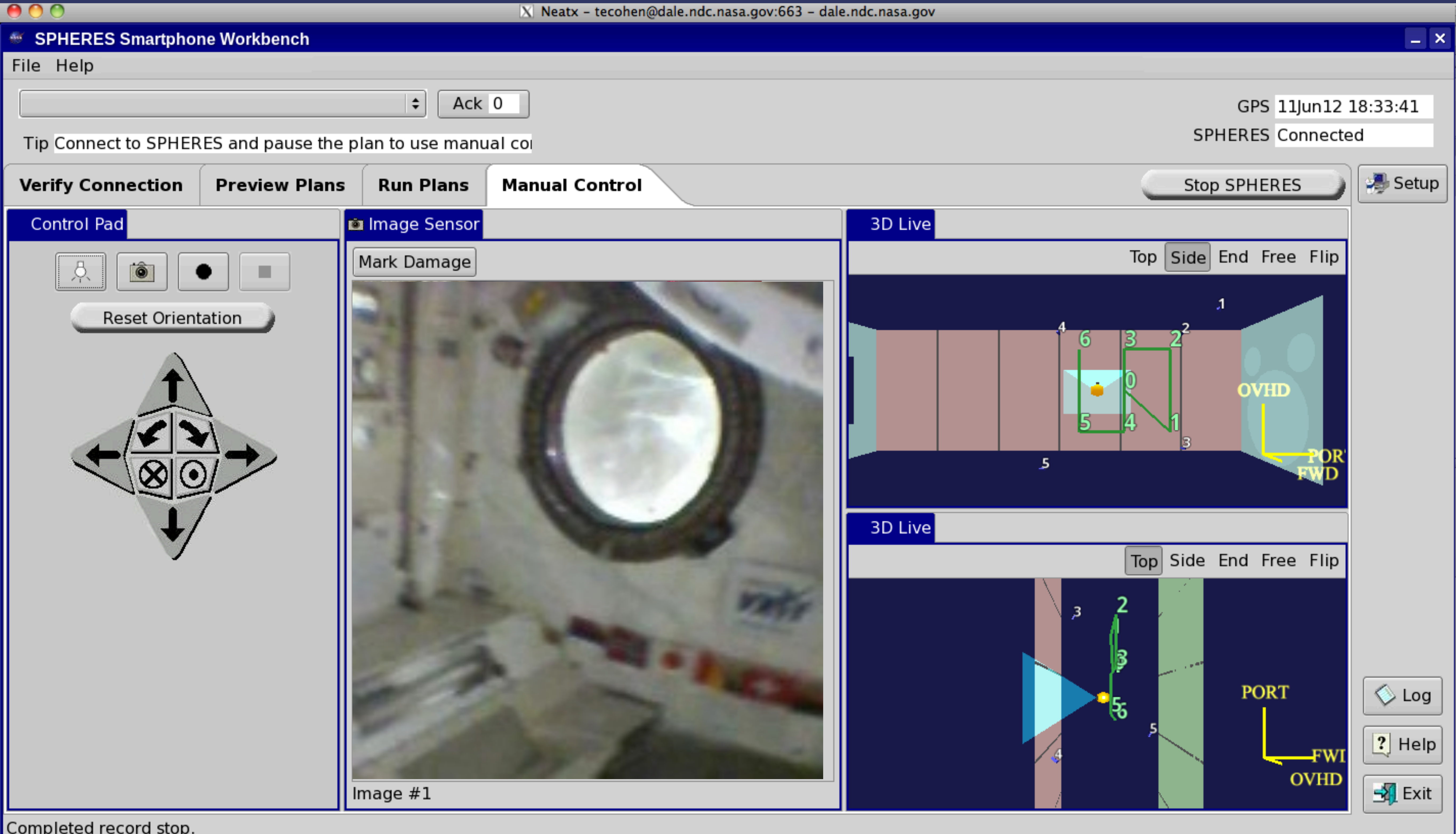
Screenshot of the SPHERES Smartphone Workbench: Previewing Plans



Screenshot of the SPHERES Smartphone Workbench: Running Plans



Screenshot of the SPHERES Smartphone Workbench: Manual Control



How to change the layout of an Eclipse RCP application

Here we are controlling the layout of the RCP application, and constructing the tabs (CTabFolder) which will control and respond to perspective switching.

```
public class IssApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {
     @Override
     public void createWindowContents(Shell shell) {
          IWorkbenchWindowConfigurer configurer =  getWindowConfigurer();
          Menu menu = configurer.createMenuBar();
          shell.setMenuBar(menu);
          shell.setLayout(new FormLayout());
          m_topToolbar = configurer.createCoolBarControl(shell);
          m_perspectiveBar = createPerspectiveBarControl(shell);
          m_page = configurer.createPageComposite(m_cTabFolder);

          m_perspectiveRegistry = configurer.getWindow().getWorkbench().getPerspectiveRegistry();
          createPerspectiveBarTabs();

          m_rightToolbar = createRightToolbar(shell);
          m_statusline = new SimpleStatusLineManager().createControl(shell);

          // The layout method does the work of connecting the controls together.
          layoutNormal();
     }
```

Do the construction and customization of the CTabFolder

```
protected Control createPerspectiveBarControl(Composite parent){
        m_cTabFolder = new CTabFolder(parent, SWT.TOP) {
                public int getBorderWidth() { return 10; }
        };
        setTabFolderFont(m_cTabFolder);
        m_cTabFolder.setMinimumCharacters(20);
        m_cTabFolder.setTabHeight(40);
        m_cTabFolder.setSimple(false);
        m_cTabFolder.setBorderVisible(true);
        m_cTabFolder.setBackground(ColorProvider.INSTANCE.WIDGET_BACKGROUND);
        return m_cTabFolder;
}
```

A method to create a tab

```
protected CTabItem createTabItem(CTabFolder tabFolder, String title,
                                 Control control, final String id) {
        CTabItem tabItem = new CTabItem(tabFolder, SWT.NONE);
        tabItem.setText("   " + title + "   ");
        tabItem.setData(id);
        tabItem.setControl(control);
        return tabItem;
}
```

A method to select a perspective

```
protected void selectPerspective(String perspectiveID, SelectionEvent e){
        IWorkbenchPage page = m_workbenchWindow.getActivePage();
        if(page != null) {
                IPerspectiveDescriptor descriptor =
                        m_perspectiveRegistry.findPerspectiveWithId(perspectiveID);
                page.setPerspective(descriptor);
                page.getActivePart().setFocus();
        }
```

Set up the tabs based on defined perspectives

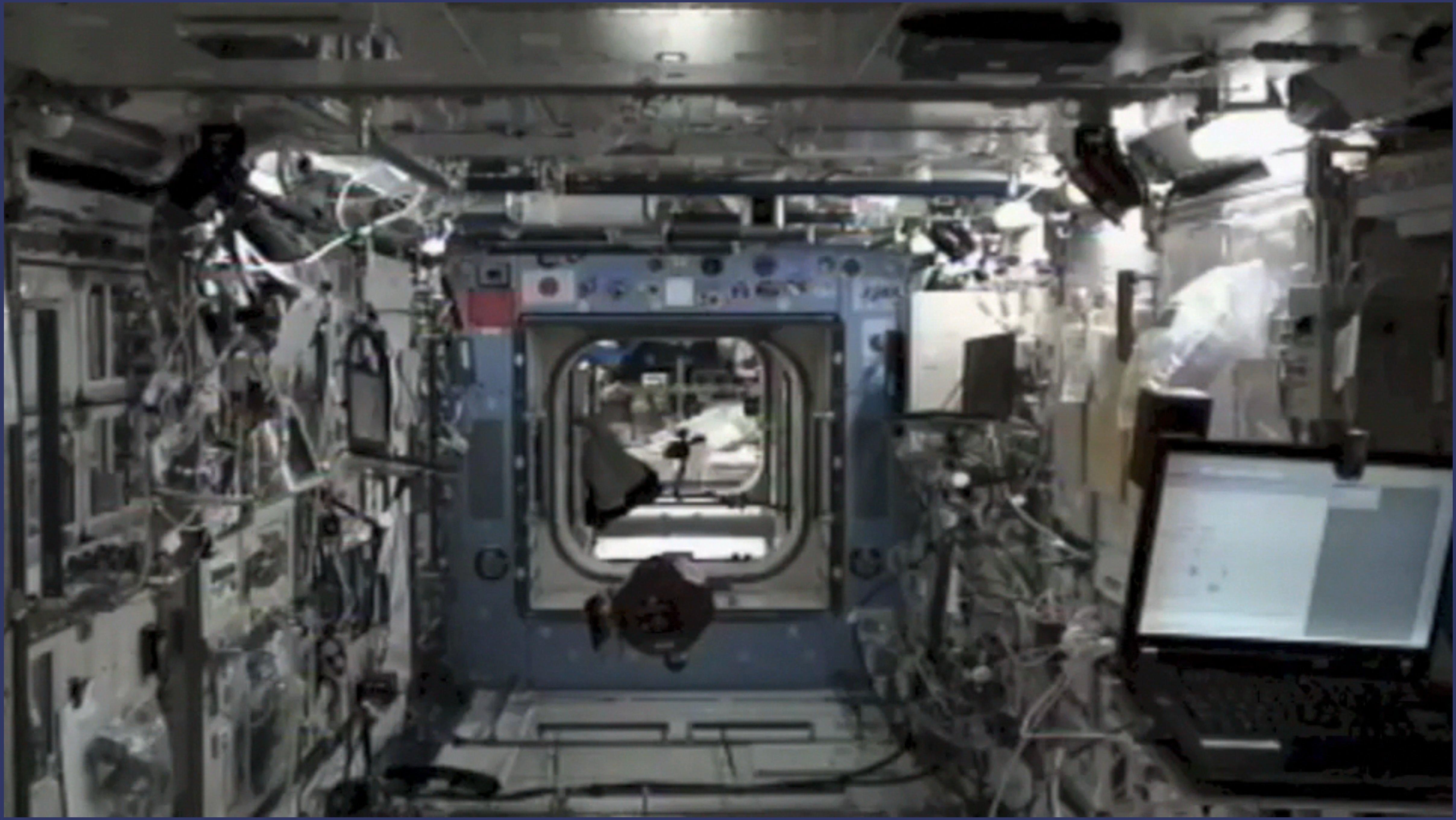
```
protected void createPerspectiveBarTabs(){
        for (String peID : getPerspectiveExtensionIds()){
                // automagically read the perspectives contributed by plugin.xml
                IConfigurationElement[] config = Platform.getExtensionRegistry().
                        getConfigurationElementsFor("org.eclipse.ui", "perspectives", peID);
                for (IConfigurationElement e : config) {
                        CTabItem item = createTabItem(m_cTabFolder,
                                e.getAttribute("name"), m_page, e.getAttribute("id"));
                }
        }


        // have the tabs listen for selection and change perspective
        final CTabFolder tabFolder = m_cTabFolder;
        m_cTabFolder.addSelectionListener(new SelectionListener() {
                public void widgetSelected(SelectionEvent e) {
                        CTabItem tabItem = tabFolder.getSelection();
                        String perspectiveID = (String)tabItem.getData();
                        selectPerspective(perspectiveID, e);
                        tabItem.getControl().setFocus();
                }
        });

        // have the tabs autochange if the perspective changes
        m_workbenchWindow.addPerspectiveListener(new PerspectiveAdapter() {
                public void perspectiveActivated(IWorkbenchPage page,
                                        IPerspectiveDescriptor perspectiveDescriptor) {
                        CTabItem foundTab = getTabForPerspective(perspectiveDescriptor.getId());
                        if (foundTab != null){m_cTabFolder.setSelection(foundTab);}
                }
        });

        m_cTabFolder.setSelection(0);
        populateTopRightButtons(m_cTabFolder);  // this is how we contribute Stop SPHERES button
        m_cTabFolder.pack();
```

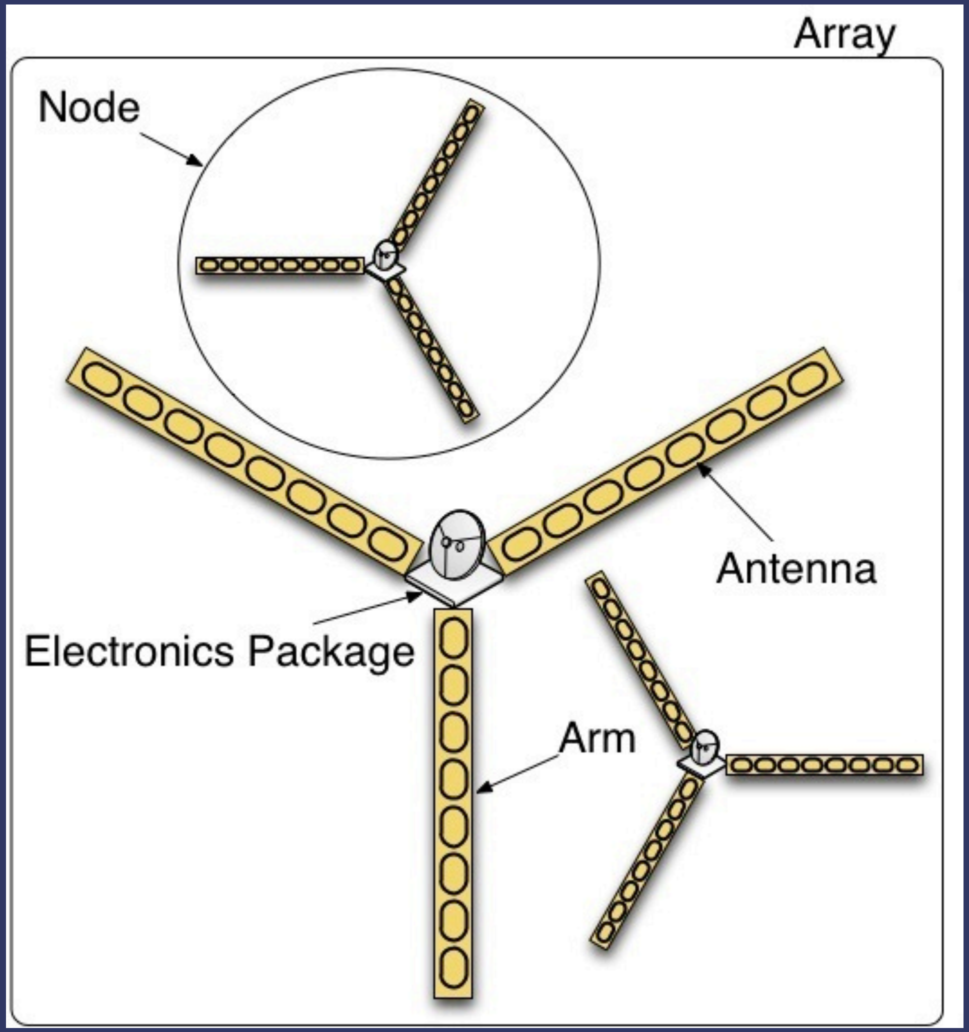
Recording of the SPHERES experiment

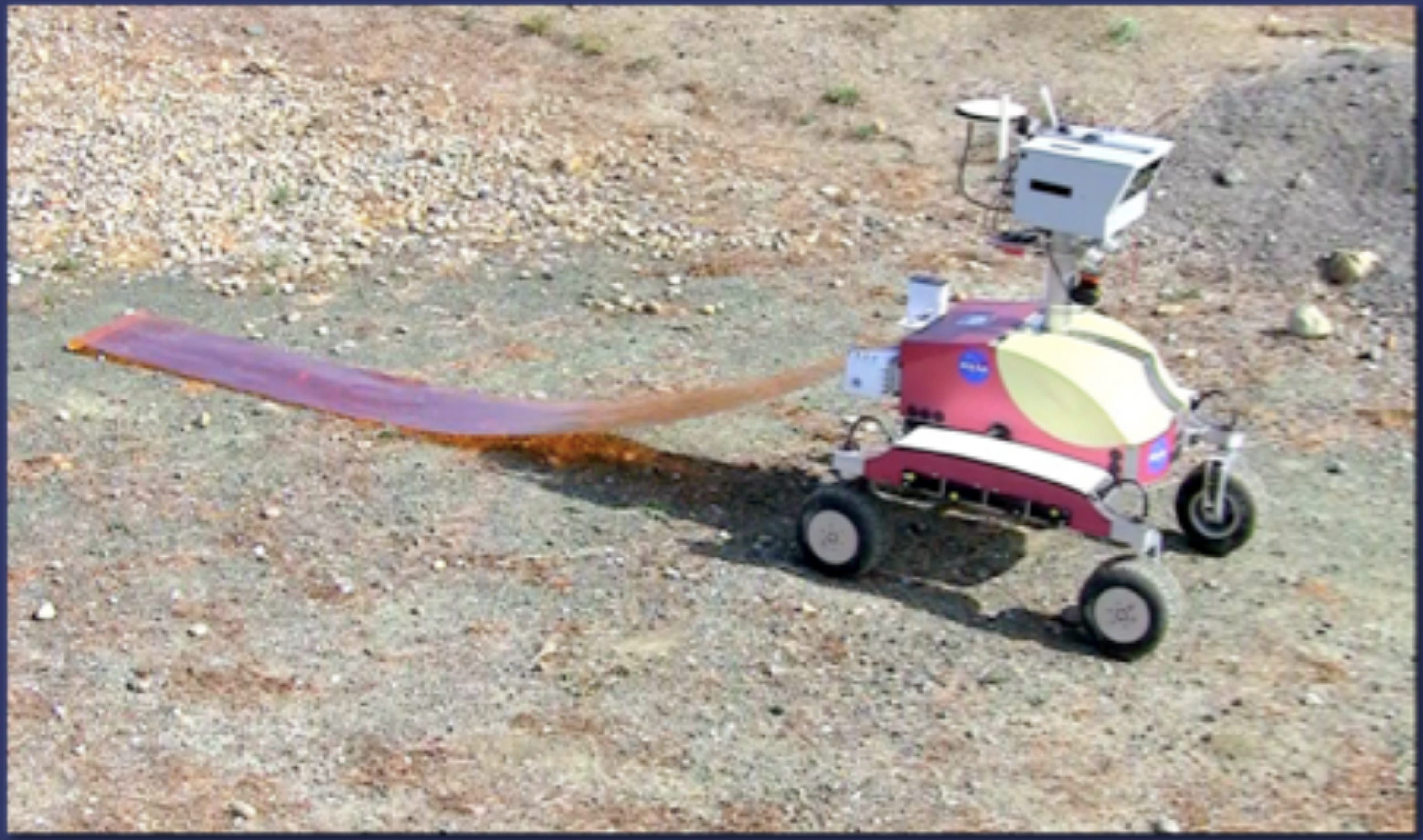


Experiment 2: Surface Telerobotics

Surface Telerobotics will examine how astronauts in the ISS can remotely operate a surface robot (K10 Rover) across short time delays. We will be simulating an astronaut teleoperating a rover on the lunar farside to deploy a low radio frequency telescope.

The telescope is comprised of three arms made of Kapton polyimide film, which will rolled out behind the rover.

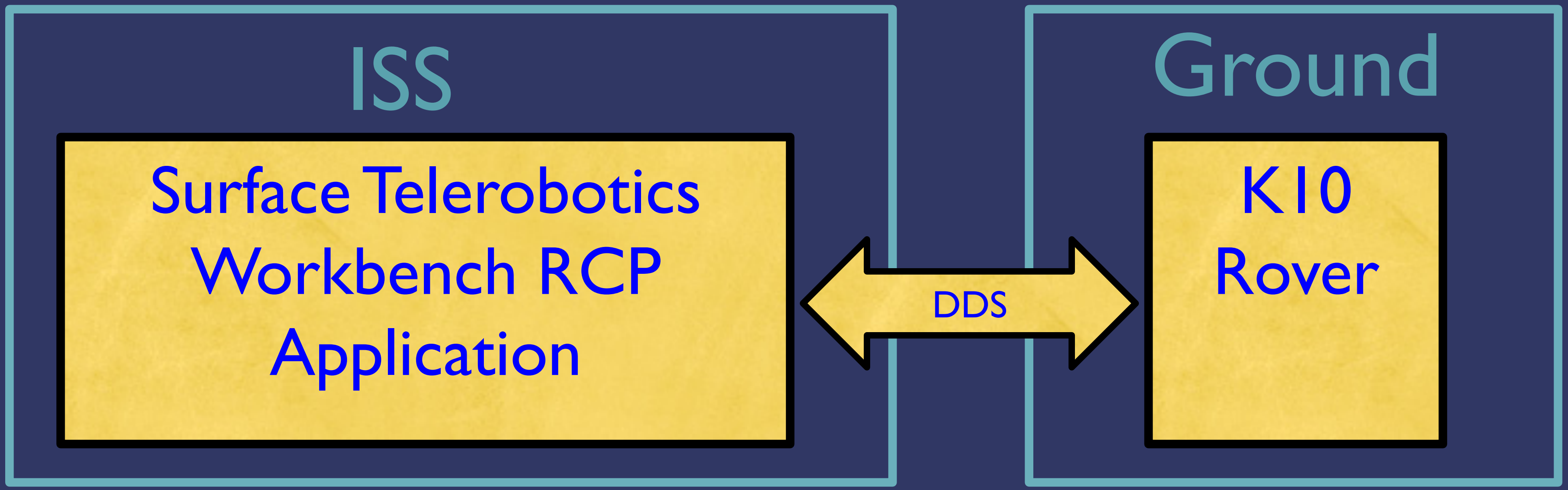




This will be the first time that an astronaut on the ISS will be controlling a sophisticated rover on Earth.

The K10 Rover has been used extensively for robotic and geologic field research. Our K10 rovers have been to numerous field sites on Earth including the Haughton Crater on Devon Island, Canada; Black Point Lava Flow, Arizona; and many sites in California.

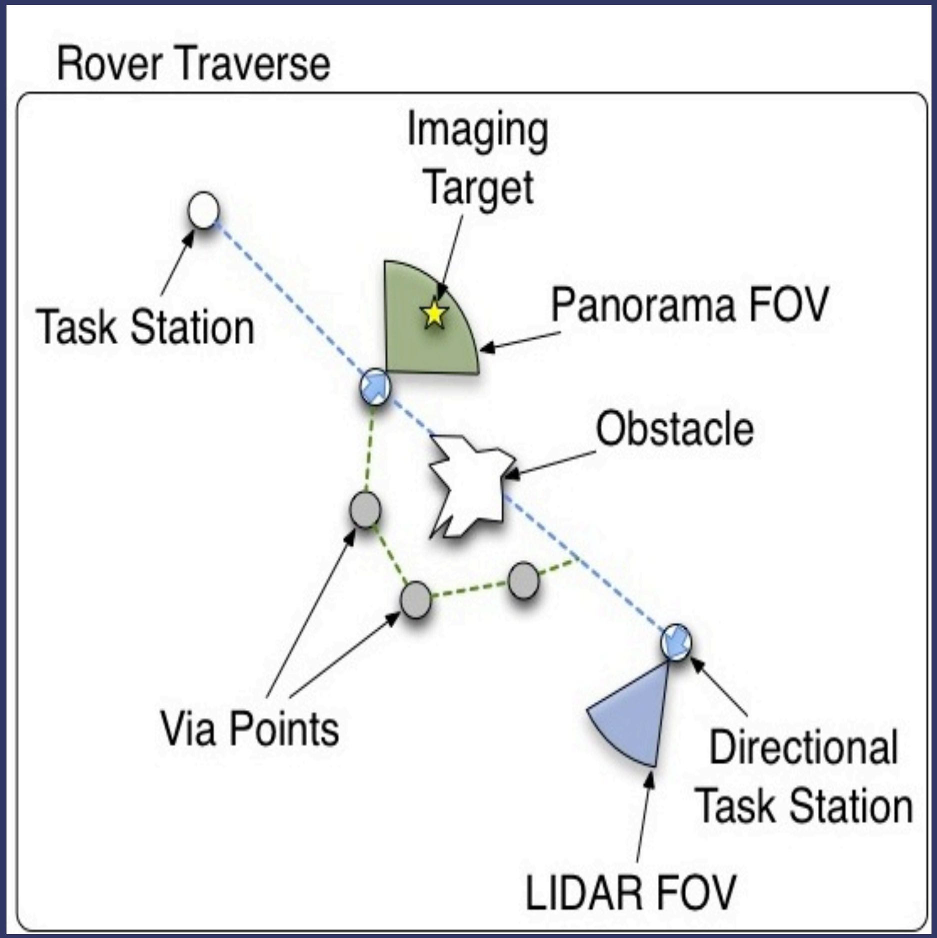
K10 has four-wheel drive, all wheel steering and a passive averaging suspension. The K10 rover's navigational sensors include a GPS System, a digital compass, stereo hazard cameras, and an inertial measurement unit. K10 rovers run Rover Software, which supports autonomous navigation and obstacle avoidance.



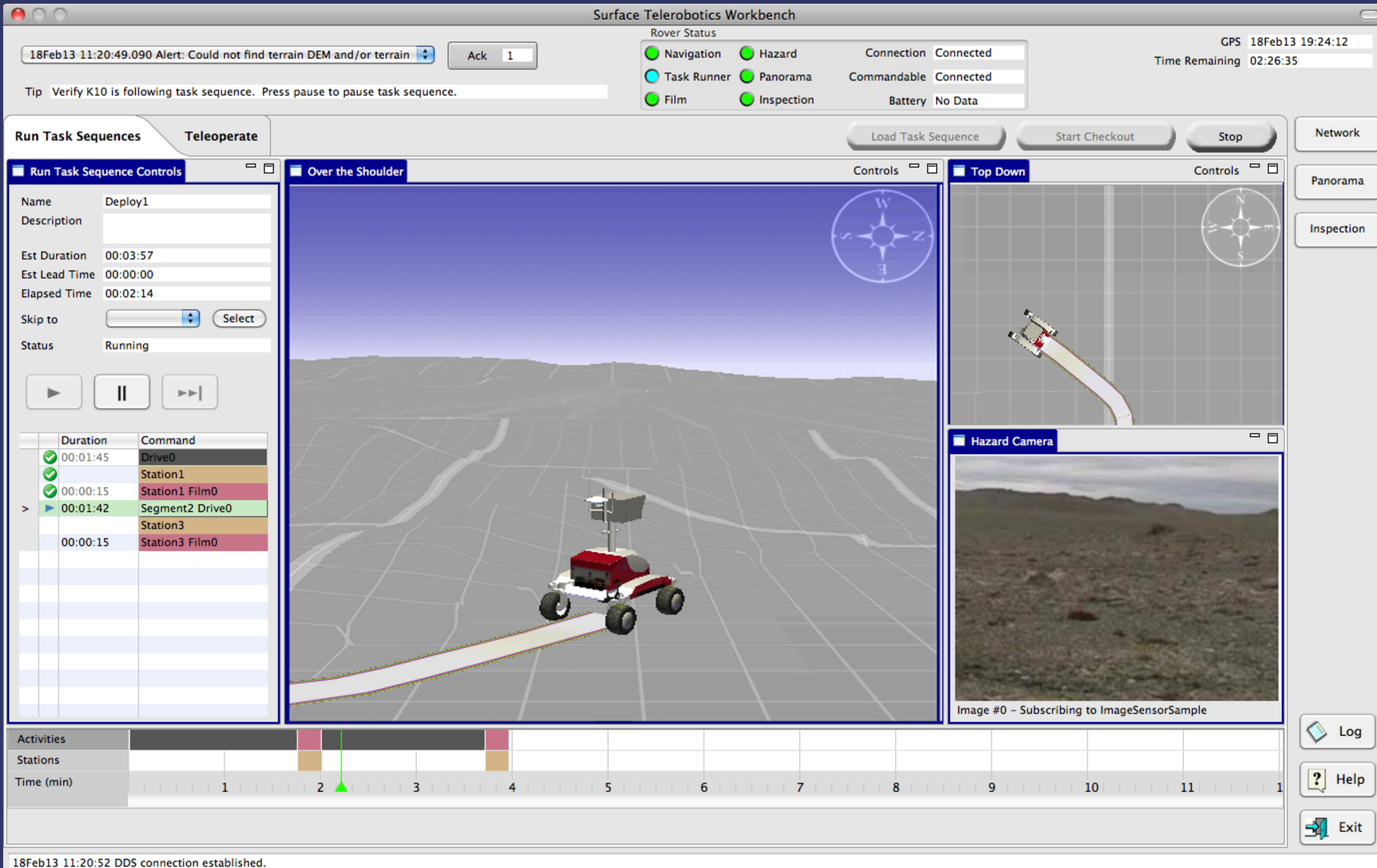
The K10 rover can be configured with different scientific instruments. For this experiment instruments include a custom panoramic camera (GigaPan), a rear-facing inspection camera to observe telescope deployment, a Velodyne to examine surface texture and to assess terrain hazards, and of course the film deployer.

We control K10 rover operations with "route plans" – a sequence of tasks that include stations, segments and tasks to do along the way. Rover software does its best to achieve the goals of the route plan, though if there is an obstacle along the way it may not succeed.

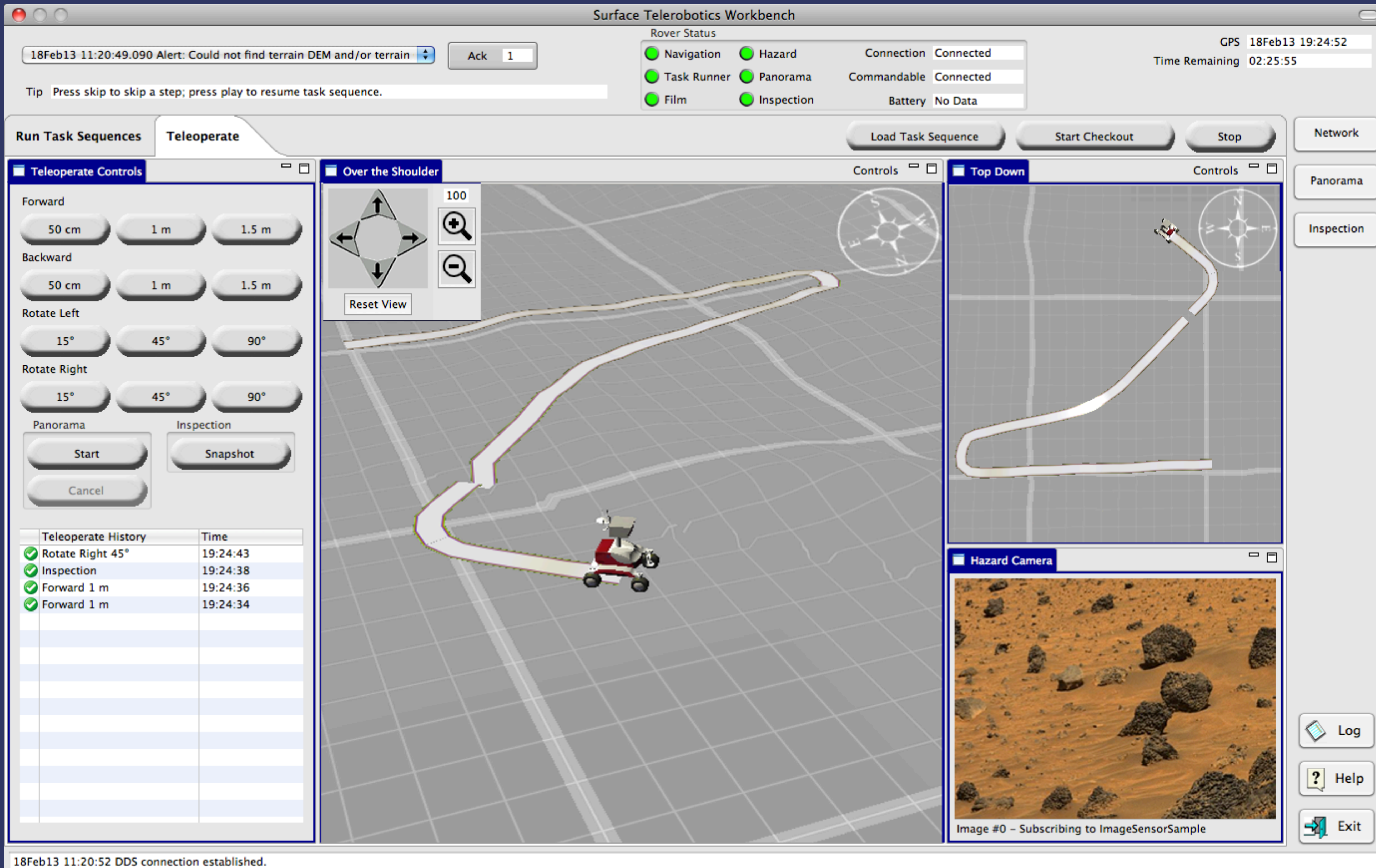
We initially developed VERVE (discussed at EclipseCon 2011) to allow rover engineers to visualize rover status in 3D within an Eclipse RCP application; the Surface Telerobotics Workbench includes some of the VERVE technology and plugins, and extends it to comply with the ISS standards.



Screenshot of the Surface Telerobotics Workbench: Running Plans



Screenshot of the Surface Telerobotics Workbench: Teleoperating the rover



How to fake buttons

ISS standards require us to create unique rounded "command" buttons, so it is clear which buttons send important commands. These buttons draw images to a graphics context, and then render text over them.

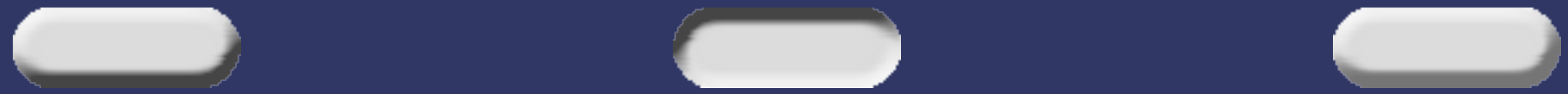
```
public class CommandButton extends Composite {

public CommandButton(Composite parent, int style) {

     super(parent, SWT.NONE);
     GridLayout gl = new GridLayout(1, false);
     gl.marginHeight = gl.marginWidth = gl.horizontalSpacing = gl.verticalSpacing = 0;
     setLayout(gl);
     m_gridData = new GridData(SWT.FILL, SWT.CENTER, true, false);
     m_gridData.widthHint = m_gridData.minimumWidth = m_width;
     m_gridData.heightHint = m_gridData.minimumHeight = m_height;
     setLayoutData(m_gridData);
     setSize(m_width, m_height);

     m_buttonLabel = new Canvas(this, SWT.NONE);
     m_buttonLabel.setSize(m_width, m_height);
     m_buttonLabel.setLayoutData(m_gridData);
     m_buttonLabel.addPaintListener(new PaintListener() {
             public void paintControl(PaintEvent e) { draw(e.gc); }
     });
```

Enabled Pressed Disabled



Add listeners to the button and set its text

```
m_buttonLabel.addListener(SWT.MouseDown, new Listener() {

     public void handleEvent(Event event) {
            if (isEnabled()){
                   m_pressed = true;
                   m_currentImage = m_pressedBgImage;
                   m_buttonLabel.redraw();
                   m_buttonLabel.update();
            }
     }});

     m_buttonLabel.addListener(SWT.MouseUp, new Listener() {
            public void handleEvent(Event event) {
                   m_pressed = false;
                   if (isEnabled()){
                          m_currentImage = m_bgImage;
                          for (SelectionListener listener : m_selectionListeners){
                                 listener.widgetSelected(new SelectionEvent(event));
                          }
                   } else {
                          m_currentImage = m_disabledBgImage;
                   }
                   if(m_buttonLabel != null && !m_buttonLabel.isDisposed()) {
                          m_buttonLabel.redraw();
                          m_buttonLabel.update();
                   }
            }
     });

 public void setText(String text){
        m_textString = text;
        draw(new GC(m_buttonLabel));
```

Draw the button

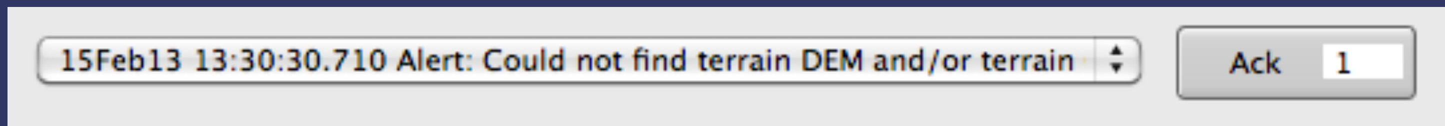
```
protected void draw(GC gc) {

        int imagey = Math.max(0, (m_buttonLabel.getSize().y - m_height) / 2);
        gc.drawImage(m_currentImage, 0, imagey);

        Color fg = isEnabled()?ColorProvider.INSTANCE.black:ColorProvider.INSTANCE.darkGray;
        gc.setForeground(fg);
        Point size = gc.textExtent(m_textString);
        int x = Math.max(0, (m_width - size.x) / 2);
        int y = Math.max(0, (m_buttonLabel.getSize().y - size.y) / 2);
        if (m_pressed){
                x +=3;
                y +=3;
        }
        gc.drawText(m_textString, x, y, true);
        gc.dispose();
}
```

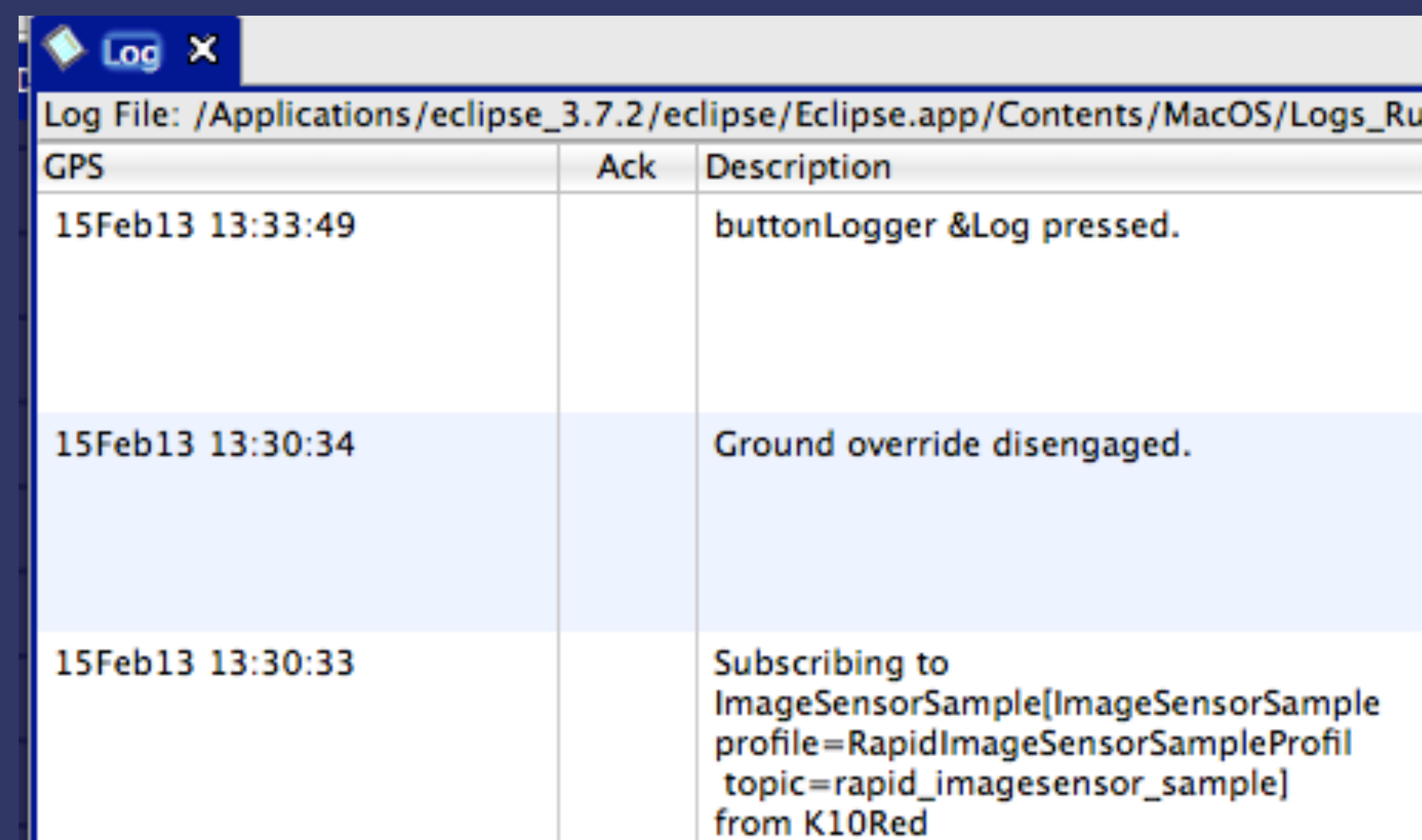
We could have gotten fancy with buttons made of multiple images which would stretch depending on the length of the text, but we didn't.

Including log4j log messages in the UI



The ISS standards require an error acknowledgement bar in a consistent place in the upper left. When important errors or alerts come in, there is an "Ack" button to the right that includes the number of unacknowledged messages.
Users can then pop up the "log" view which shows the time, ack state and description of messages that came in.

We use Apache's log4j framework to log messages, and OSGi's LogListener to read the messages in and display them in ourLogView.



How we get log4j messages into our Log View

In our code, we just call logger.error("message")

```
public class IssLogView  extends ViewPart implements LogListener {

public IssLogView() {
            setReader(IssLogService.getInstance().getLogReader());
            m_comparator = new LogViewComparator(); // this lets us sort the way we want

            // add the logger appender to the Apache Log4j framework
            Logger.getRootLogger().addAppender(new IssLoggerAppender());
            Logger.getLogger(IssLoggerAppender.class.getName()).setAdditivity(false);
     }

protected class IssLoggerAppender extends AppenderSkeleton {
     protected void append(LoggingEvent event) {
            if (event.getLevel().isGreaterOrEqual(MIN_LEVEL)){
               String status = getEventLevelString(event.getLevel()) +
                                                    event.getRenderedMessage() ;
               String ds = LogViewUtils.convertToCorrectDateFormat(event.getTimeStamp());

               // convert this log message from the file to our IssLogEntry class,
               // and contribute it to the view to display in the table.
               processEntry(event.getLevel().toInt(), ds + " " + status);
               asyncRefresh(true);
            }
     }
}
```

```
// simple class to hold log entries
public class IssLogEntry {

     protected String m_time;
     protected Level m_level;
     protected String m_description;
     protected boolean m_ack;
```

Intelligent Robotics Group
at NASA Ames Research Center

- K10 Rover among others
- SPHERES
- xGDS Ground Data Systems
- VERVE 3D within Eclipse
- Contributed the moon to Google Earth
- Mars-o-vision (mars.planetary.org)
- GigaPan robotic camera
- GeoCam disaster response
- Ames Stereo Pipeline
- Vision Workbench
- Tensegrity research

… and more!

http://irg.arc.nasa.gov