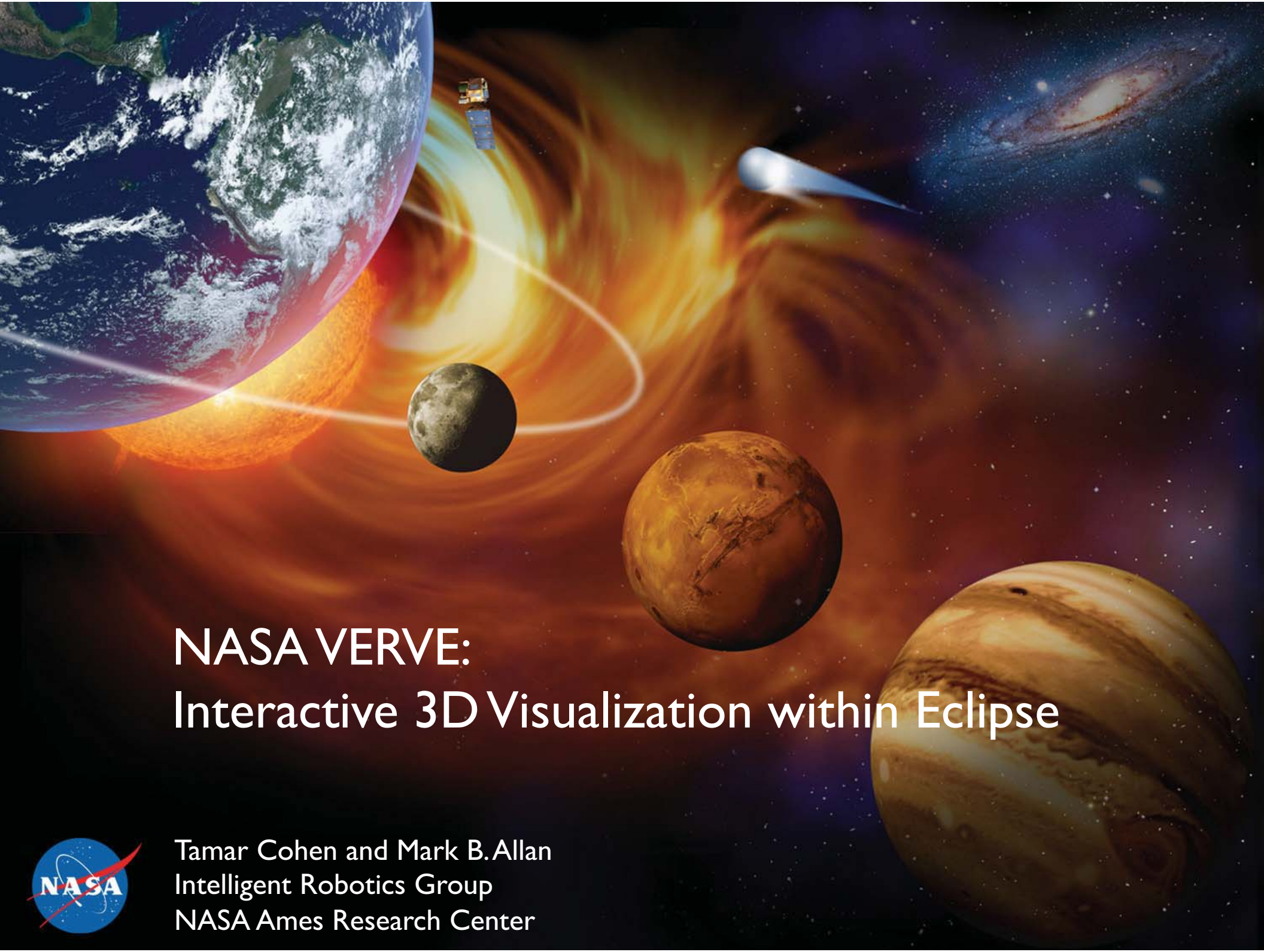


NASA VERVE:
Interactive 3D Visualization within Eclipse

Tamar Cohen and Mark B. Allan
Intelligent Robotics Group
NASA Ames Research Center

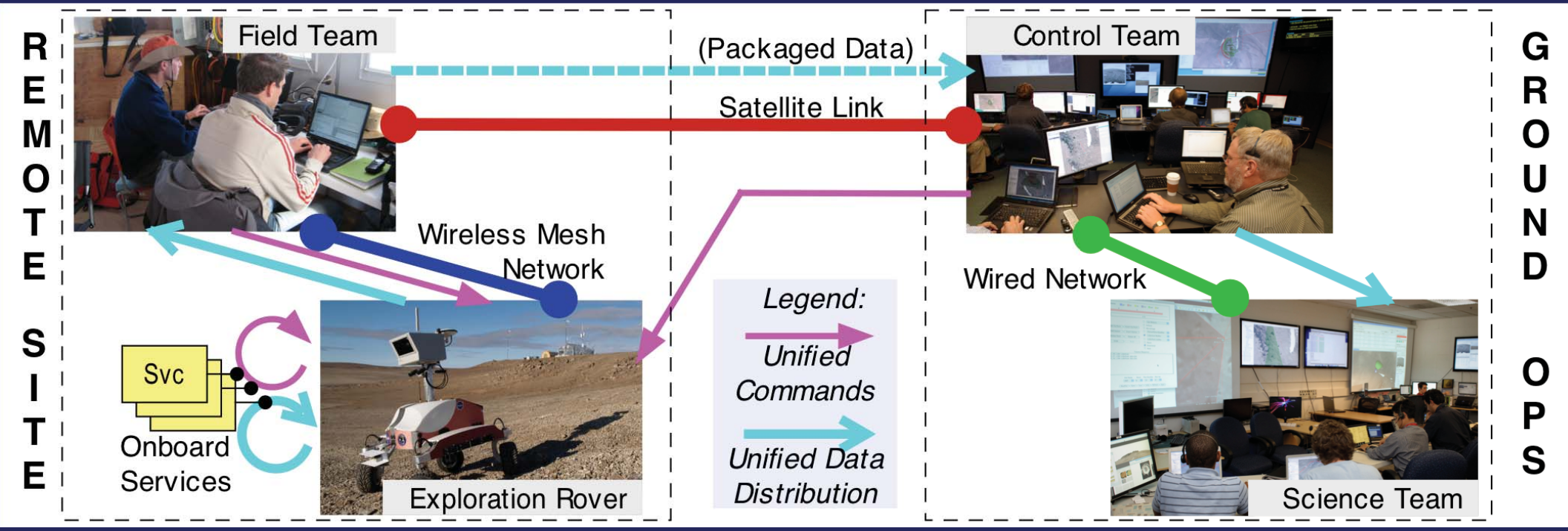
The Intelligent Robotics Group (IRG) at NASA Ames Research Center develops robotic rovers as research platforms for autonomous navigation, human robot interaction, and robot software.

Often we send a small team of scientists and robot engineers into the field with some rovers, and science and engineering backroom teams remotely operate the rovers.

As such, it is necessary to visualize where the rover is, what algorithms it is executing, and what scientific data it is gathering.

K10 Red at Haughton Crater, Devon Island CA

Concept of Operations



We have created a number of tools to assist with remote situational awareness. Some of these are web tools, and quite a few are Eclipse RCP applications, based around VERVE.

VERVE is an integrated 3D view built on top of Ardor3D.

Flight Control Room at Ames during HMP experiment

Science backroom at Ames

We develop and work with many different robots for different needs, and attach various scientific instruments to them. We also collaborate with different NASA centers to support remote operation of their robots.

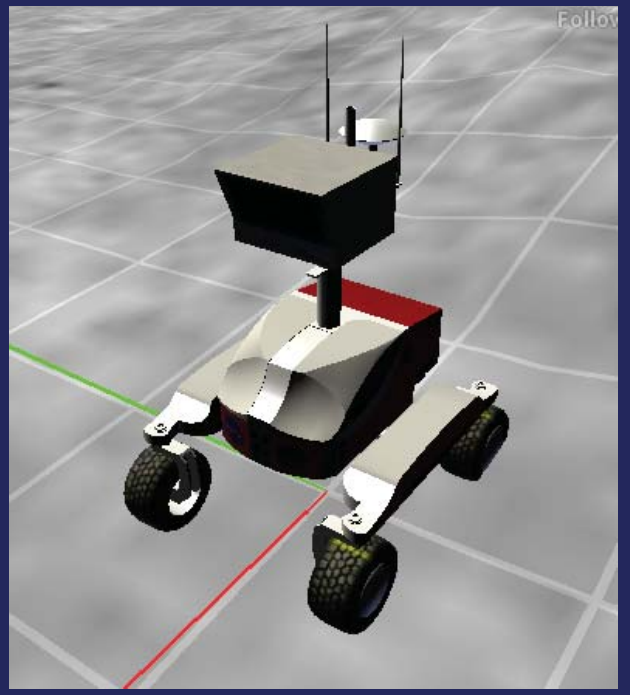
IRG's K-series rovers have four-wheel drive, all wheel steering and a passive averaging suspension. Navigational sensors include a GPS system, digital compass, stereo hazard cameras, and an inertial measurement unit. K-series rovers run Rover Software, which supports autonomous navigation and obstacle avoidance.

Krex at Basalt Hill, CA

The VERVE 3D View is an Eclipse view with contents rendered by the Ardor3D libraries which we have wrapped in a plug-in.

Ardor3D provides user interface support within the view, eg the compass rose and overlay text.

Ardor3D provides hooks for mouse control and keyboard input, along with typical controls for a 3D graphics library (ie cameras, scene, lighting, etc). It supports standard 3D file formats.



A simple model of K10Red; we keep the models small and efficient and only articulate what will give us useful information.

The Ardor3D scene graph is comprised of
- Spatials to define geometry and rendering settings
- Nodes spatials with parents and children

For each type and instance of a robot, we create a RobotNode (extends Node) which contains nodes that represent its model (3D object) and child nodes for representation of scientific data.

We have a reference between our conceptual representation of a robot and each of its parts, and the Ardor3D nodes which represent each concept.

```
public class RobotNode extends Node {

    final AbstractRobot m_robot;

    RobotModelNode m_model;
    Node           m_concepts;
    Node           m_sensors;
    Node           m_base;

    public RobotNode(AbstractRobot robot, Node model) {
        super(robot.getName());
        m_robot = robot;
        setRobotModel(model);
    }
```

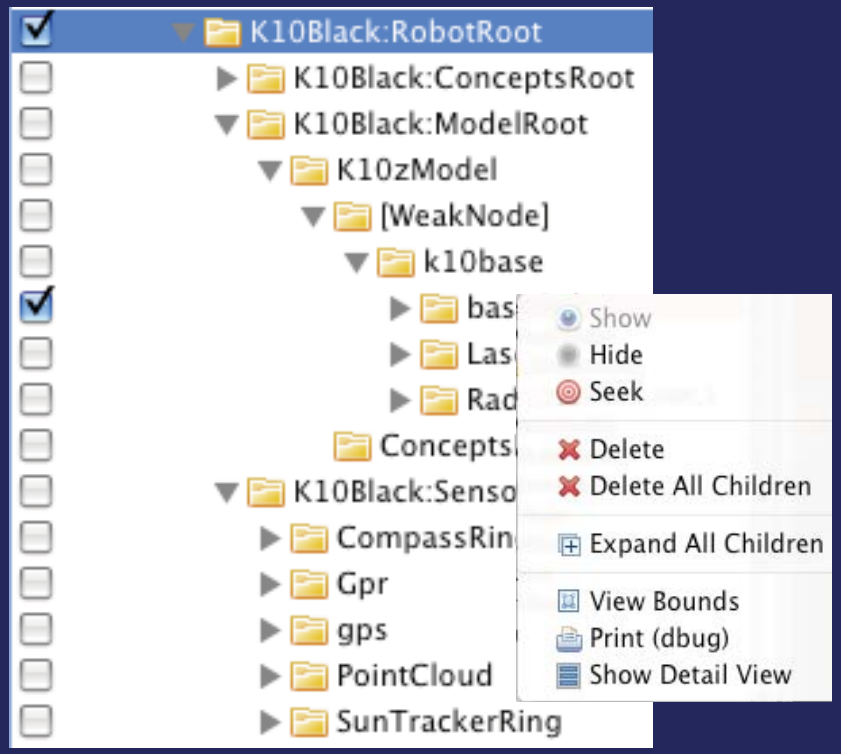
We have created a simple interface to connect the incoming telemetry to the robot parts that are rendered:

```
/**
 * interface for scenegraph -> telemetry glue for robot visualization parts
 */
public interface IRobotPart {
    public String getPartName();
    public AbstractRobot getRobot();

    public void connectTelemetry() throws TelemetryException;
    public void disconnectTelemetry() throws TelemetryException;

    public void attachToNodesIn(Node model) throws TelemetryException,
IllegalStateException;

    public void handleFrameUpdate(long currentTime);
    public void reset();

    public boolean isVisible();
    public void setVisible(boolean state);
}
```

Desert RATS featured several robots, vehicles, and astronauts with sensors

A model of a space suit with instrument backpack and the Centaur 2 rover, which we worked with at Desert RATS

We have a standard Eclipse view which includes a tree populated with the contents of the scene graph. This tree can be extremely deep.
Since elements are dynamically added to and removed from the 3D scene, the tree must be populated with WeakReferences to support garbage collection.



Scene Graph View

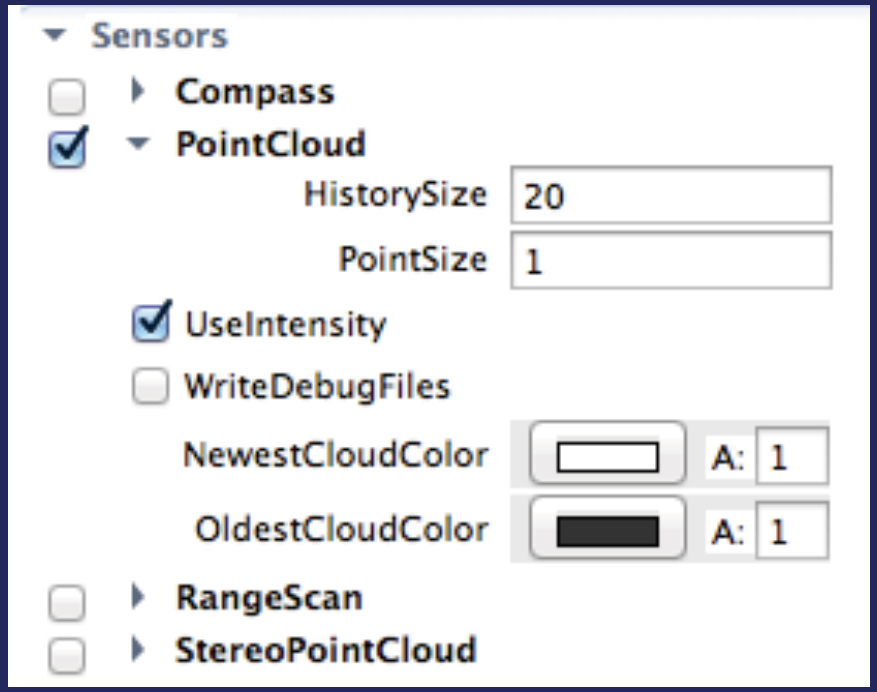
When various events happen, the tree refreshes asynchronously.

Checkboxes show and hide 3D models in the Ardor3D view by setting their render state.

```
if (show){
      spatial.setCullHint(CullHint.Inherit);
} else {
      spatial.setCullHint(CullHint.Always);
}
```

We use Java reflection and SWT databinding to automatically generate SWT UI components based on the Java objects defined in the robot class; manipulating these widgets affects how the components are rendered in the Ardor3D scene.

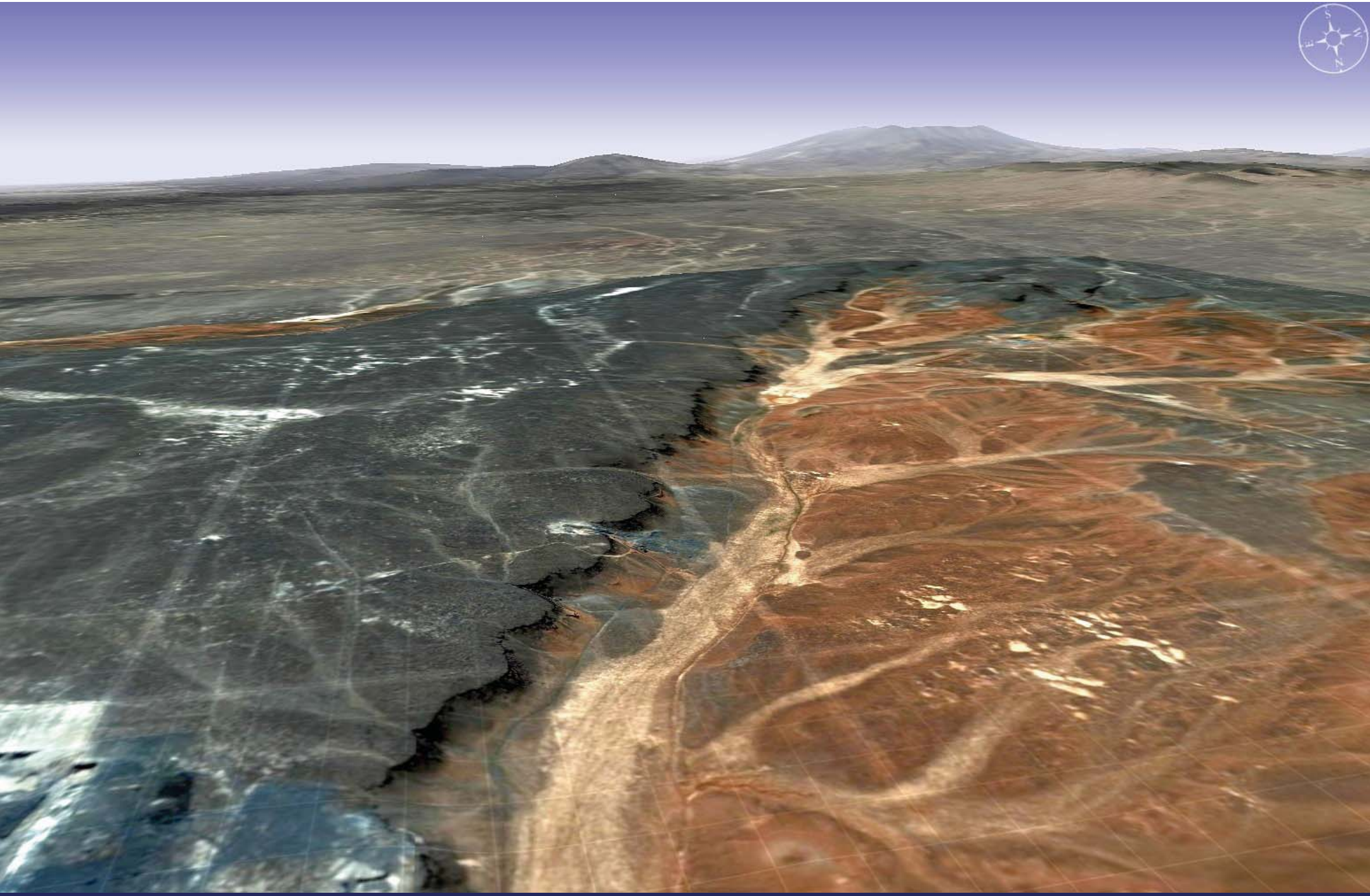
This allows our robot engineers and scientists to customize the visualization based on what they are looking for.



Typically we work with rovers to explore terrain that has not been well mapped. We may have satellite imagery and information about the terrain. This data is at a low resolution and may not be accurate. As the rovers traverse the area, they build more accurate maps.

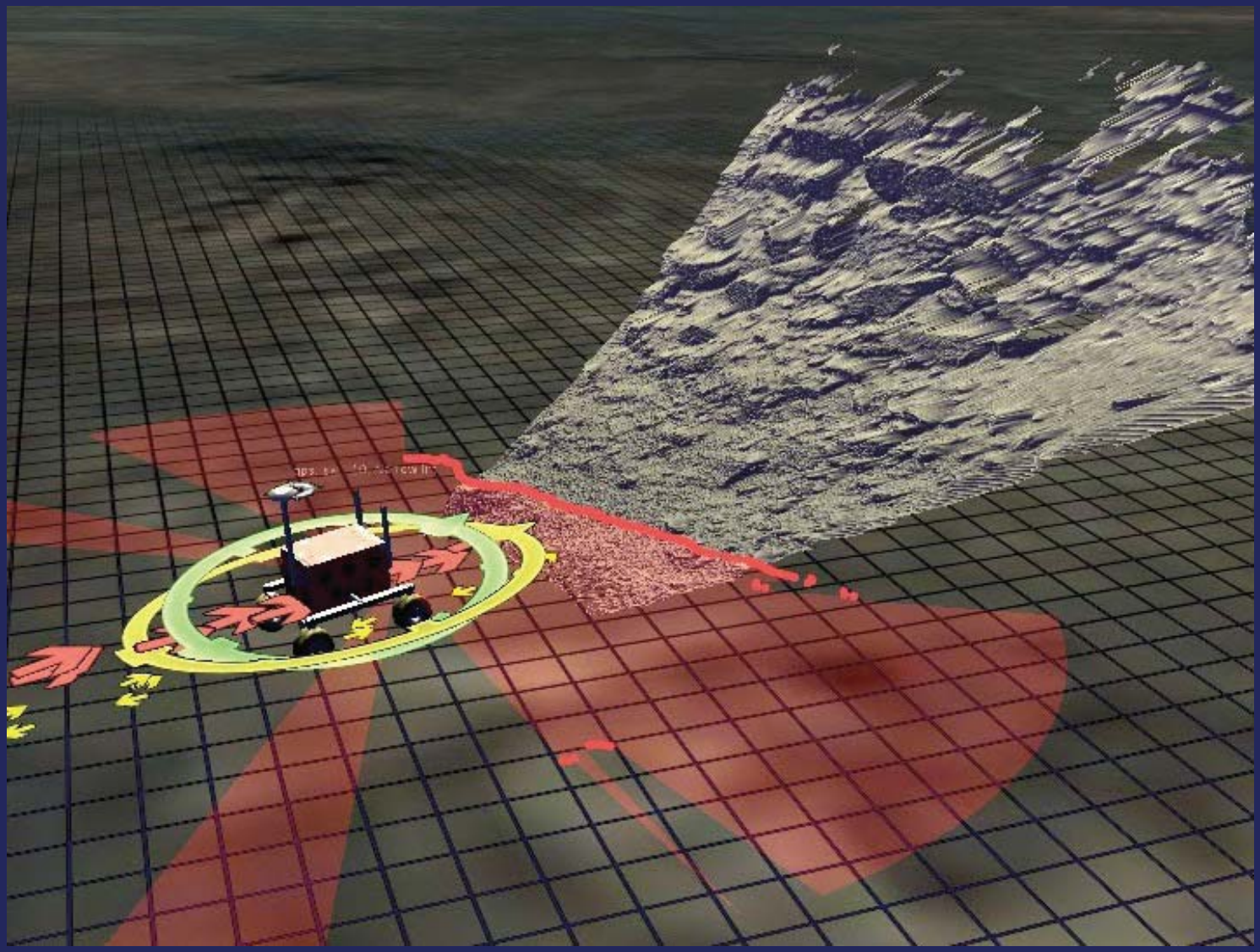
We use LIDAR, a remote sensing technology that measures distance with a laser. We use LIDAR both for a line scan and to return a point cloud. We also do stereo reconstruction with our two hazard cameras.

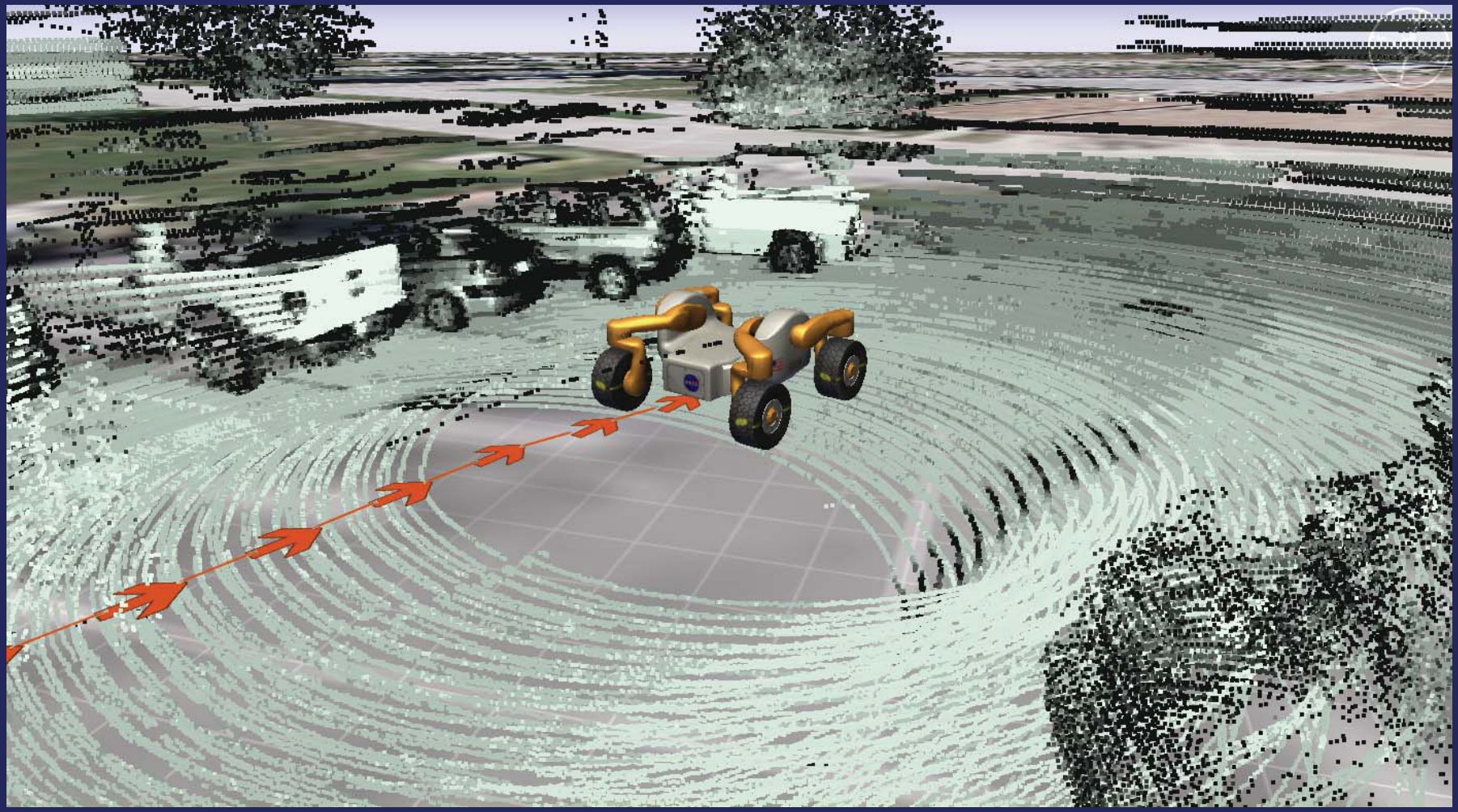
When we get this information back to our Eclipse RCP application from the rover, we can "ground truth" or adjust the terrain to the discovered data.



Black Point Lava Flow as a GeoTIFF with levels of detail, rendered with haze

Representation of LIDAR terrain in VERVE



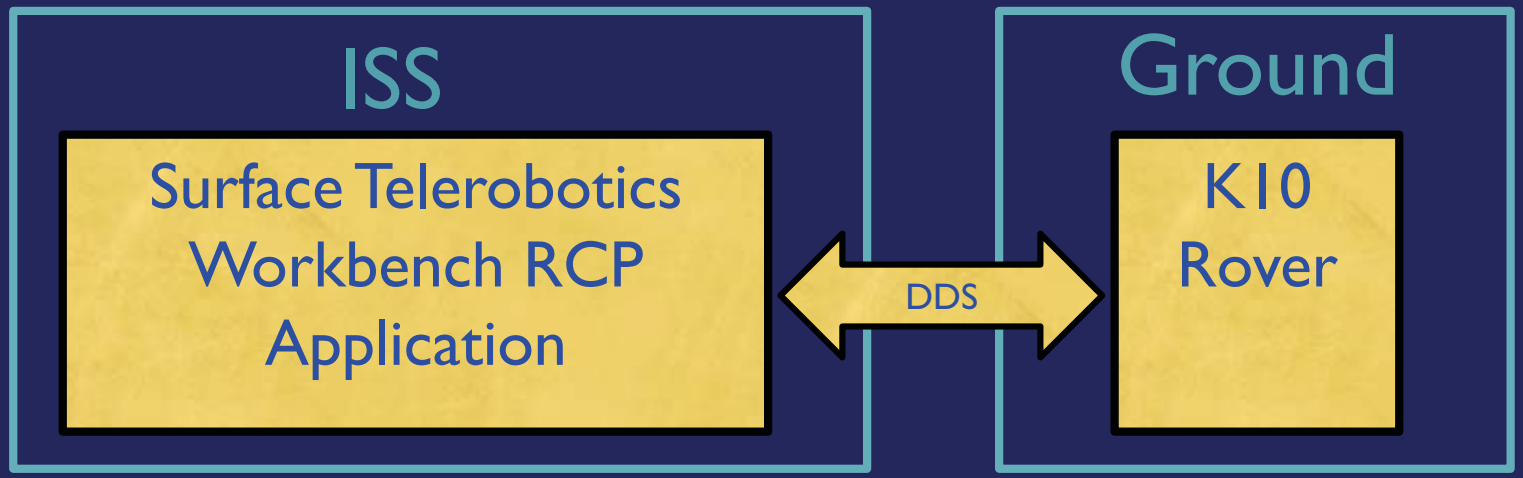


You can see the line scan and point cloud data that Centaur 2 has captured as it drives across a parking lot. Red arrows show the path it has taken.

We use DDS, the Data Distribution Service for Real-Time Systems as our middleware. By having this standard, we can create IDL (Interface Definition Language) files that describe the data which is being serialized between publishers and subscribers. We define data formats for commands and for telemetry, and various consumers (robots, software applications) can subscribe to the published topics.

We configure DDS to handle poor connectivity and dropouts, since we are simulating space missions which will have these issues.

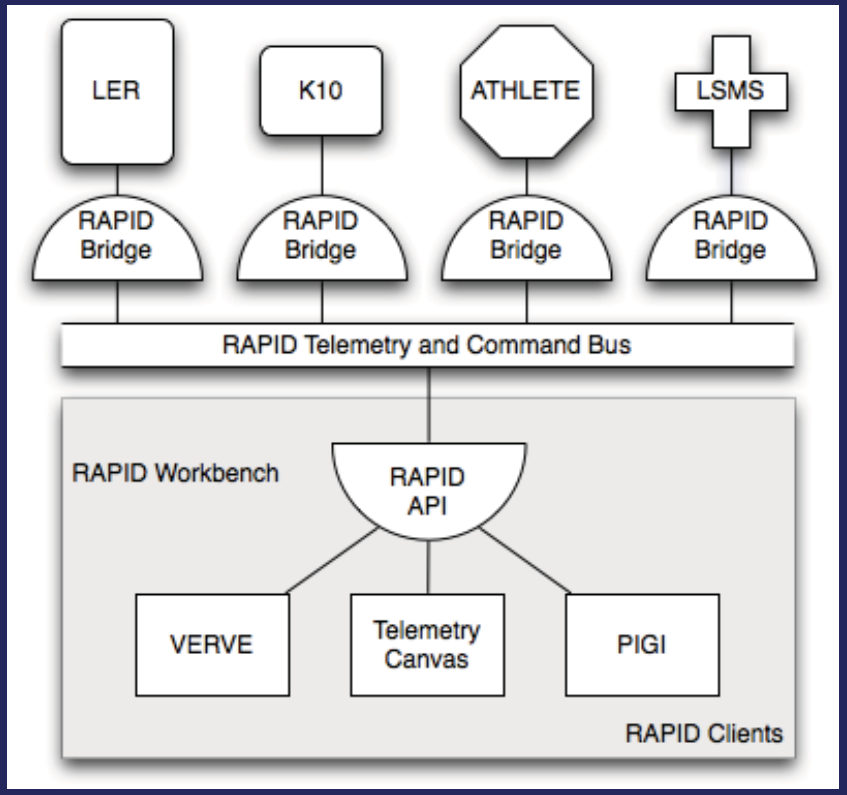
Adhering to the DDS standard makes it more straightforward to integrate with various robots and scientific instruments.



We used DDS to have astronauts on the International Space Station control our K10 Rover at NASA Ames

To facilitate collaboration between NASA centers, we came up with a standard called RAPID, built on top of DDS. This includes a set of IDL files and Java libraries for common commands and telemetry.

http://rapid.nasa.gov/



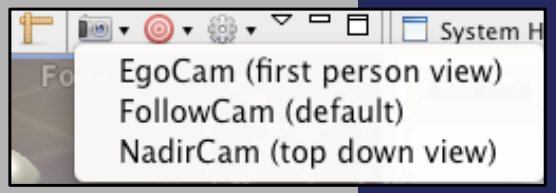
For our RAPID VERVE implementation, we have a base class to connect robot parts to RAPID telemetry.

```
public abstract class RapidRobotPart extends AbstractRobotPart implements
IRapidMessageListener
{
 …
 @Override
 public void connectTelemetry() throws TelemetryException {
    if(isTelemetryEnabled()) {
      try {for(MessageType msgType : rapidMessageTypes()){
      RapidMessageCollector.instance().addRapidMessageListener(getParticipantId(),
                                    getRapidRobot().getAgent(), msgType, this );
```

This way, when the telemetry comes in, the relevant data can be cached and when it is time to rerender, handleFrameUpdate is called.

We have data coming in too fast to render, so we throttle that back:

```
timer = new Timer(); // an Ardor3D timer which uses System.nanoTime
    frameHandler  = new FrameHandler(timer);
    frameHandler.addUpdater(new LogicalLayerUpdater());

    // we expose camera controls in the Eclipse UI
    frameHandler.addUpdater(new CameraControlUpdater());
    renderUpdateThread = new RenderUpdateThread(applicationPlugin,
                                                frameHandler, timer);
    renderUpdateThread.start();
```



Our very simple render update thread runs as follows:

Once the system and display are ready (until the display is disposed)

Check the time that the last change occurred;

if the elapsed time is enough, asynchronously run an update

```
frameHandler.updateFrame();
```

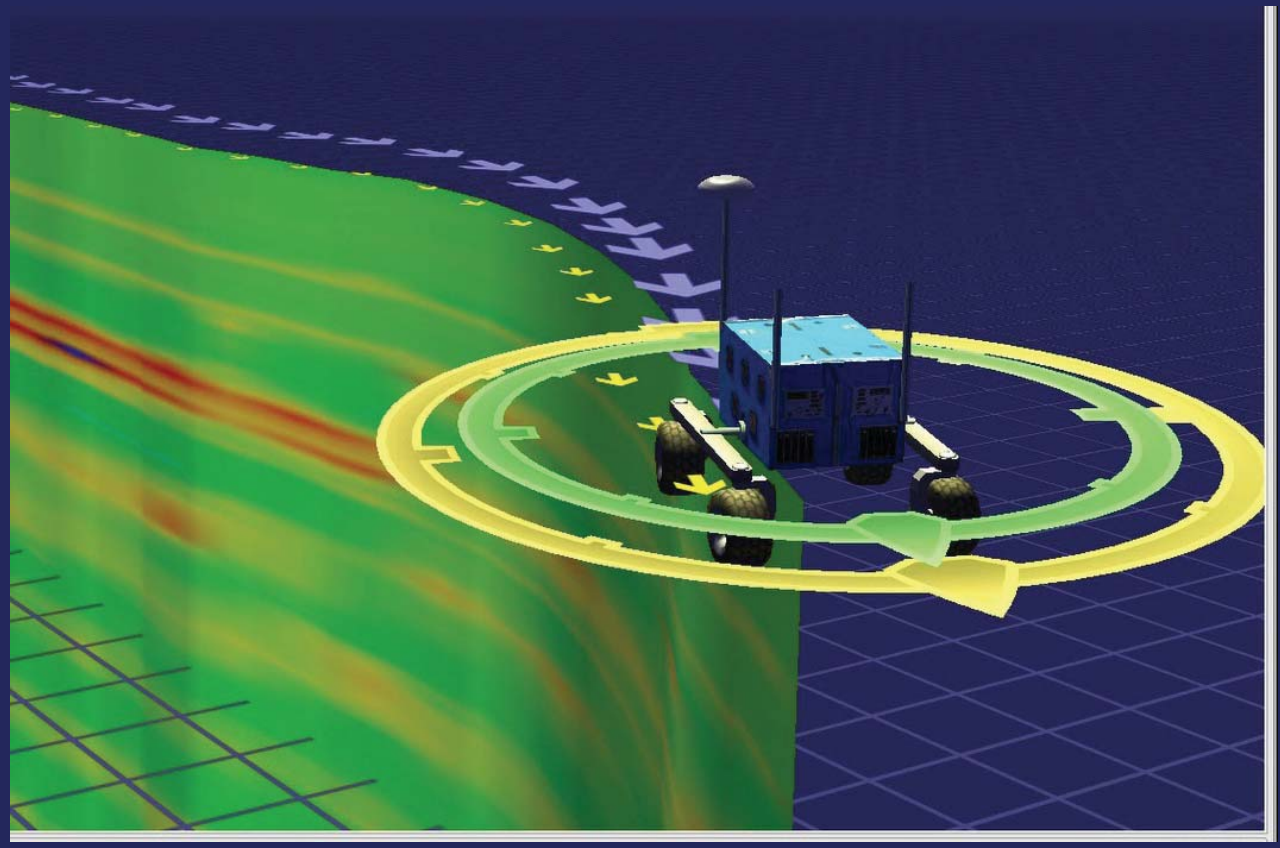
Sample handleFrameUpdate for joint data

```
public void handleFrameUpdate(long currentTime) {
    if(isDirty()) {
        try {
            float[] ja = m_angleData;
            JointInfo joint;
            for(int i = 0; i < m_joints.length; i++) {
                if(i < m_angleData.length) {
                    joint = m_joints[i];
                    if( joint.spatial != null && !Float.isNaN(ja[i]) ) {
                        m_rot.fromAngleNormalAxis(ja[i]+joint.offset,
                                                  joint.rotAxis);
                        joint.spatial.setRotation(m_rot);
                    }
                }
            }
        }
        catch(Throwable t) {
            logger.log(t);
        }
        setDirty(false);
    }
}
```

Scientific Visualization

Here we are rendering data coming back from a Ground Penetrating Radar (GPR) system, so scientists can analyze information about the substrate. They can customize the rendering of the GPR telemetry.

Arrows indicate previous position, and the rings indicate the sun tracker and the magnetic compass.

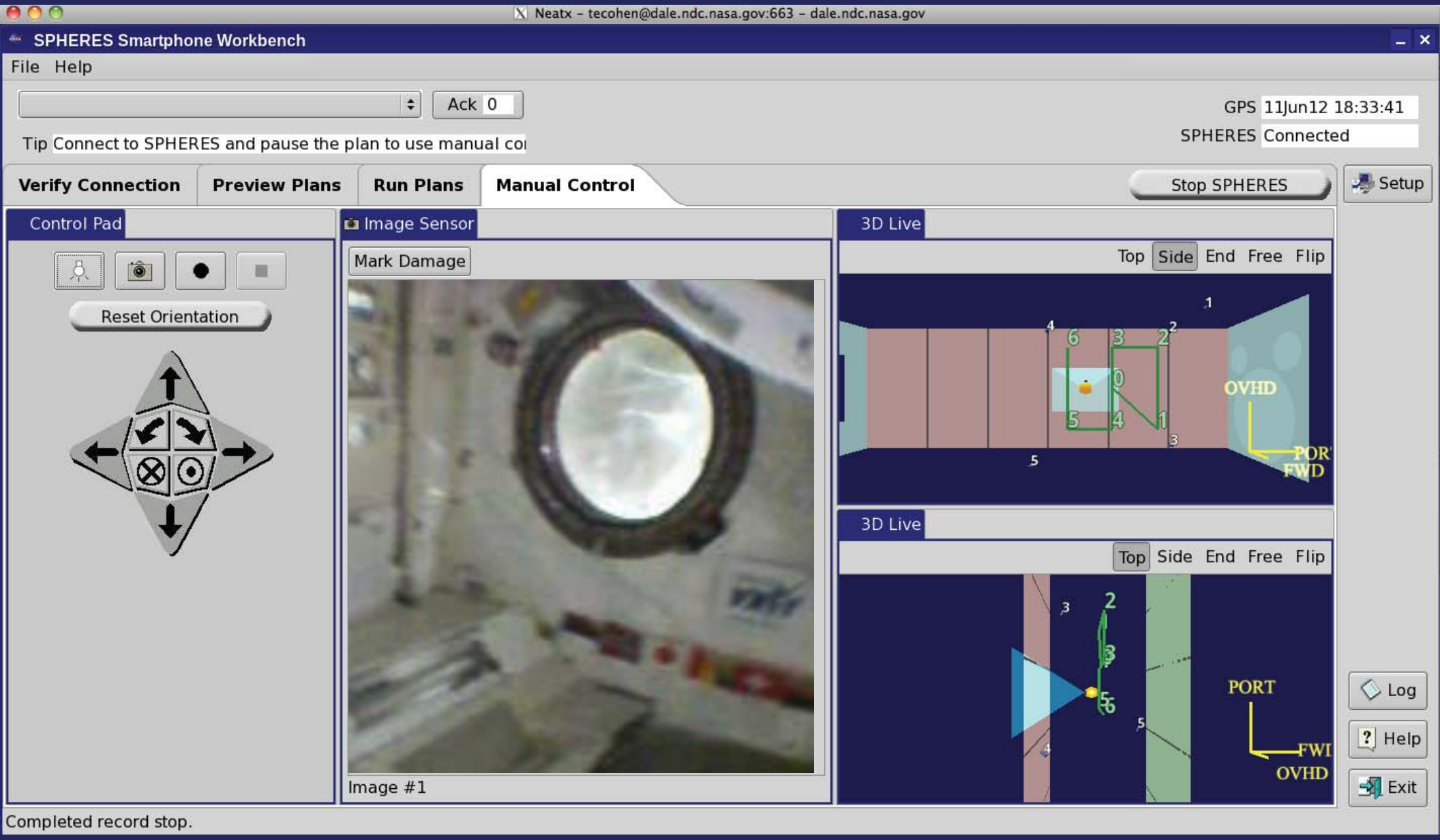


We also work with the SPHERES on the International Space Station (ISS); we run DDS on an Android Smartphone which we connect to the SPHERES, and view and control it from our Eclipse RCP Application.

In our experiment in 2013, we simulated an internal inspection of an ISS module .

We have commanded it from the ground and astronauts have commanded it from the ISS.

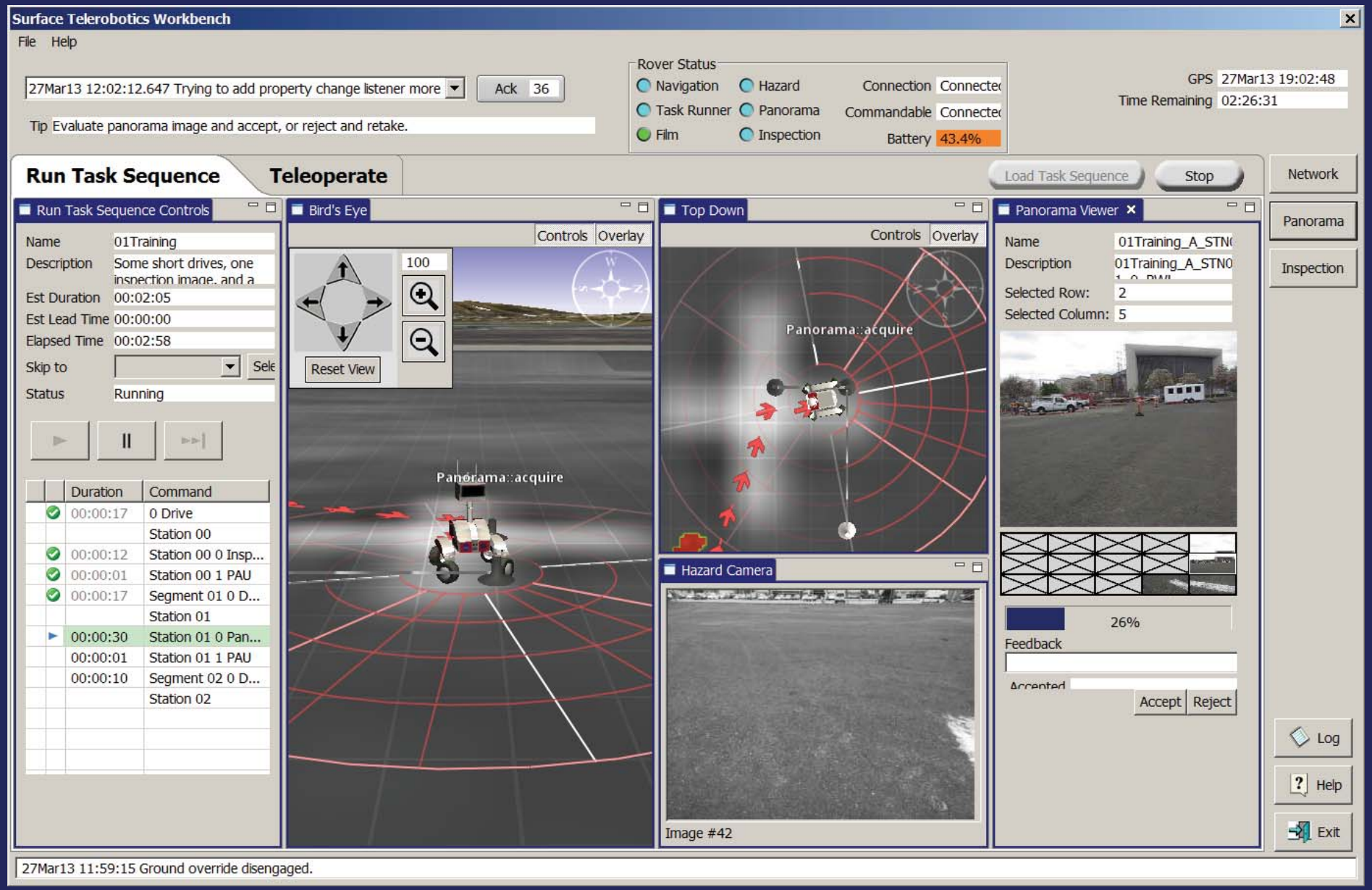
Screenshot of the SPHERES Smartphone Workbench: Manual Control



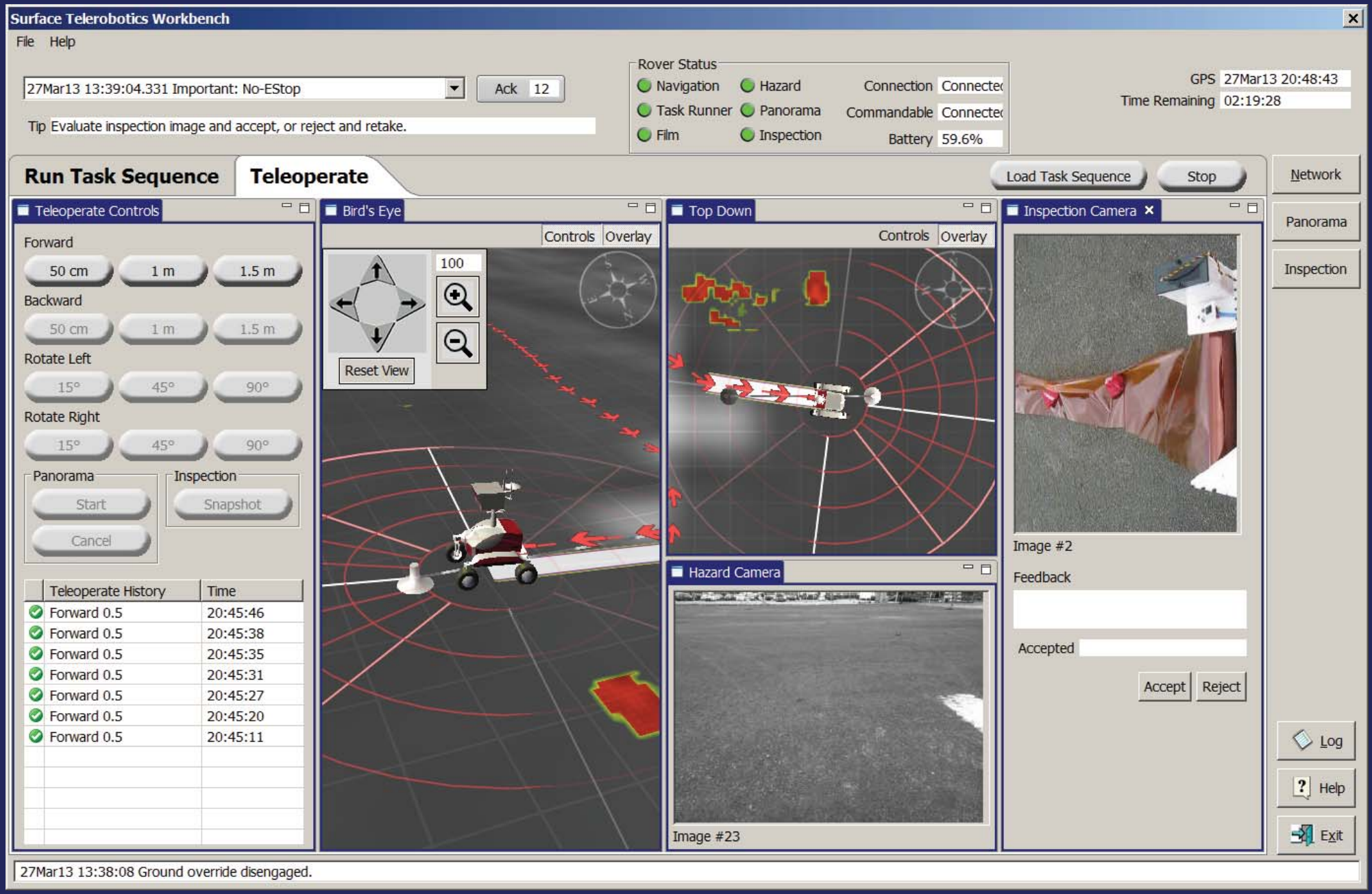
Deployment of a Telescope

The Surface Telerobotics experiment in 2013 examined how astronauts in the ISS can remotely operate a surface robot (K10 Rover) across short time delays. We simulated an astronaut teleoperating a rover on the lunar farside to deploy a low radio frequency telescope.

Rover running a Route Plan; Panorama coming in

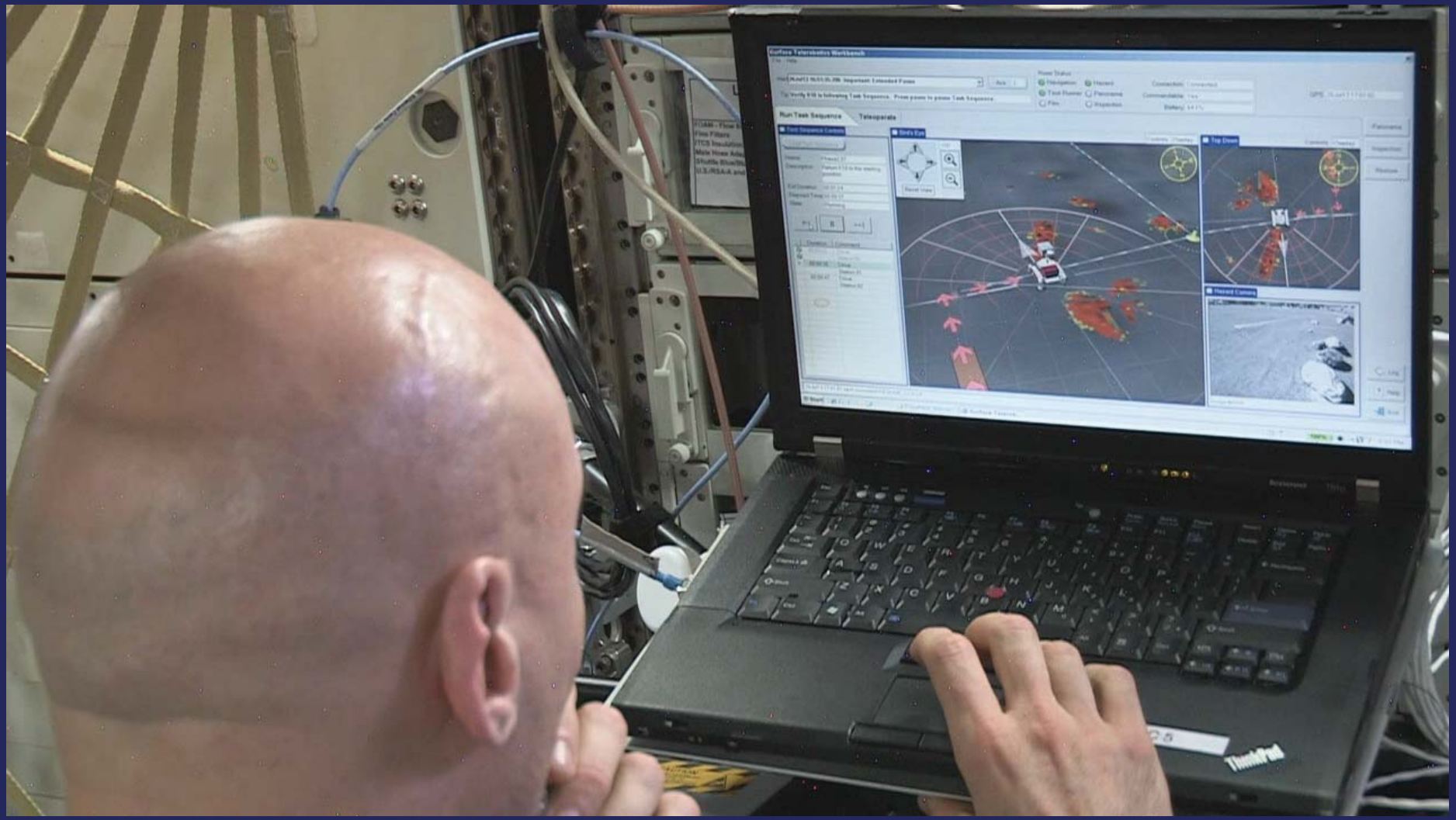


Manually moving the rover forward and inspecting the film



During our Surface Telerobotics experiment, we worked with a 3rd party company (TRACLabs) to subscribe to DDS messages providing informational alerts about the state of our rover. We displayed these in a growl-like overlay in the VERVE 3D View.

Surface Telerobotics Experiment in action: Luca Parmitano on the ISS operating K10 at Ames



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

http://sourceforge.net/projects/irg-verve/