Introduction to OpenROV and Trident
Hardware and Electronics Overview
Software Overview
Platform Extensibility
Video Delivery and Recording With DDS
Bridging the Gaps: Where DDS Has Room to Grow
Conclusion - Q&A
About OpenROV

Started in 2012 by co-founders Eric Stackpole and David Lang...

ERIC STACKPOLE
Co-founder, CEO
Lifelong tinkerer, MS Mechanical Engineering
Spacecraft mechanism designer, NASA Ames Research Center
2014 White House “Champion of Change”
Antarctic expedition ROV engineer and pilot

DAVID LANG
Co-founder, President
Writer: MAKE: Magazine, Popular Mechanics, etc.
Author: Zero to Maker
Member of NOAA’s Ocean Exploration Advisory Board
TED Senior Fellow, National Geographic Explorer

...who together created the world’s first affordable underwater robot kit, the OpenROV DIY Kit:
Their Mission?
To empower everyone to explore the world’s oceans and waters
OPENROV DIY KITS HAVE CREATED A GLOBAL COMMUNITY.
The Team

ERIC STACKPOLE
Co-founder, CEO

COLIN HO
Product Design Engineer, Roboticist
MS Mechanical Engineering (UC Berkeley)
NASA JPL, NSF Graduate Research Fellow
Team lead for MSLED Antarctic ROV

CHARLES CROSS
Embedded Systems Developer
BS Electrical Engineering (VCU)
NASA LaRC
Architect at NASA Autonomy Incubator

BRIAN ADAMS
Software Team Lead
BS Computer Science (Univ. of Oklahoma)
RelayHealth, VP Product Development
Avid web backend developer

GILBERT MONTAGUE
AI and Computer Vision Researcher
Physics (Baldwin-Wallace Univ.)
NASA LaRC, GRC, KSC
Developer, NASA Autonomy Incubator

ZACK JOHNSON
Sales, Project Manager
BS Electrical Engineering (Clemson Univ.)
TechShop, San Francisco
Led development of TechShop storefront

MARGARET SINSKY
Operations Manager
BA Humanities (Seattle University)
Yelp, Account Executive
Customer relationships, BizDev

ERIC STACKPOLE
Co-founder, CEO

DAVID LANG
Co-founder, President

BRIAN GRAU
Mechanical Engineer, Project Manager
BS Mechanical Engineering (Santa Clara University)
SCU Robotic Systems Laboratory
Led winning MATE ROV competition team

WALT HOLM
Electrical Engineer
BS, MS Electrical Engineering (MIT)
USAF, EE Consultant
Amateur Archaeologist

BRIAN GRAU
Mechanical Engineer, Project Manager
BS Mechanical Engineering (Santa Clara University)
SCU Robotic Systems Laboratory
Led winning MATE ROV competition team

Pasha Reshetikhin
Electrical Technician
Avid electronics hobbyist
Easy-to-use, ready-to-fly
- Robust, reliable, extensible
- Depth rated to 100m
- Price point: $1,699
- Raised $815k on Kickstarter
Trident in Action
Trident Use-Cases

AQUACULTURE

INSPECTION

OIL & GAS

RESEARCH

SALVAGE/MARINE

SEARCH & RESCUE
Trident Use-Cases

- AQUACULTURE
- INSPECTION
- OIL & GAS
- RESEARCH
- SALVAGE/MARINE
- SEARCH & RESCUE

But also...
- Diving
- Fishing
- Citizen Science
- Education
- And more...
Hardware and Electronics Overview

- Hardware
  - Vehicle
  - Topside & Tether
  - Client Devices and Platforms
- Electronics: What’s Inside
Vehicle

- Abrasion and impact resistant bumpers
- Precision thrusters
- Low-drag design (speeds of 2 m/s)
- LED lights (600 lumens)
- HD Camera (wide angle, 1080p)
Vehicle

Vertical Thruster

Rear Thrusters

Fast & Maneuverable
Trident can keep up with strong currents or stay in place underwater.

Goes to 100m
Trident can go deeper and stay down longer (up to 4 hours) than divers.

Lightweight & Portable
Weighing only 3.4 kg (7.5 lbs), Trident fits easily in a backpack or as carry-on luggage.

Maintenance Free
A freshwater rinse after each dive keeps Trident flying smoothly.
Topside & Tether

- Custom Tether
  - Neutrally Buoyant (Freshwater)
  - Kevlar Reinforced
  - Two stranded copper wires
  - Designed for Homeplug AV
Topside & Tether

- Custom Tether
  - Neutrally Buoyant (Freshwater)
  - Kevlar Reinforced
  - Two stranded copper wires
  - Designed for Homeplug AV

- Custom Waterproof Connectors

- Topside acts as Wireless AP
  - 2x2 MIMO
  - 802.11n
  - Status Lights
  - USB Extension Port
The Whole Package

❖ Durable, Waterproof Carrying Case ➢ Fits in Carry-On Luggage

❖ Tether Reel ➢ For managing longer tether lengths
Client Devices and Platforms

- Currently supports modern Android devices
  - Android 5.0+
  - Supports standard USB and Bluetooth Controllers
  - iOS support planned in near future
- Prototype VR Support
  - Using GearVR or Google Cardboard
- Desktop support for R&D clients

Custom Control Device
Vehicle Onboard Processors

**CPU**
- RPI3
- 1.2Ghz Quad Core
- Memory
- 10/100M Ethernet
- 32GB+ uSD
- 2GB RAM

**General Purpose MCU**
- Atmel SAMD21
- Cortex-M0+
- 256KB Flash
- 32KB SRAM

**Motor Control MCUs**
- 3X Cortex-M0
- 32KB Flash
- 8KB SRAM
MCU Sensors & Subsystems

Port Motor → Atmel SAMD21 → External Pressure → UART
Vertical Motor → Atmel SAMD21 → External Temperature → UART
Starboard Motor → Atmel SAMD21 → Internal Pressure → UART

Atmel SAMD21 → Tail Status Lights → I2C
Atmel SAMD21 → Forward Lights → GPIO
Atmel SAMD21 → Battery Fuelgauge → PWM
Atmel SAMD21 → 9DOF IMU → UART

UART
I2C
GPIO
PWM
RPI3 Sensors & Subsystems

- ROVLink (Homeplug AV)
- Ethernet
- I2C
- GPIO
- USB
- Raspberry Pi 3
  - Humidity Sensor
  - Internal Temperature
  - Configuration EEPROM
  - SWD Switch
  - Power Failure Warning
  - Factory Reset Switch (Hall Effect)
- Camera
Communication & Sensing Challenges

- Electromagnetic waves don’t propagate well through water
  - No GPS
  - No WiFi
  - Many standard sensing methods don’t work as well, if at all
Taking a closer look at the visible spectrum...
Communication & Sensing Challenges

So what are the options?

❖ Acoustic
  ➢ Propagates easily over long distances
  ➢ Expensive
  ➢ Very low bandwidth

❖ Optical
  ➢ Highly expensive
  ➢ Limited bandwidth
  ➢ Limited to moderate range

❖ Tethered
  ➢ Generally high bandwidth
  ➢ Different tether types and wired protocols have different capabilities and costs
    ■ Fiber Optic
    ■ Conductive
  ➢ Entanglement hazards
  ➢ Affects control of vehicle
OpenROV Employs Homeplug AV Over Twisted Pair

- Bandwidth of ~80Mbps at the MAC layer
  - Latest Homeplug standards support 500Mbps to 1Gbps
- Acts as an ethernet bridge
- Only needs two wires
- Range tested up to ~350m
- Cheap hardware solution
  - Based on Atheros PLC Chipset
  - Packaged into a PCI Express form factor
Software Overview

- Core Vehicle Services
  - trident-core
  - update
  - beacon
  - camera
- Build and Deployment Process
Core Services: trident-core

Functions:

- Acts as a bridge between the MCU databus and DDS databus
  - Each sensor/actuator treated as a subsystem
  - Data transmitted over UART using Mavlink protocol
- Implements lightweight reliability protocol
Core Services: update

Functions:
- Listens for version information from the MCUs that gets emitted by core
  - Additionally checks flash CRC validation, in case of corruption
- Allows user to decide when to apply updates

- RPI3 selects chip with address pins
- Bit-bang SWDIO/SWCLK over GPIO
- OpenOCD used for SWD operations
  - Flash
  - Verify
  - Dump
  - Debug

Advantages
- No bootloaders required
- No external memory
- Debug access to all devices
Core Services: beacon

❖ Functions:
  ➢ Emits vehicle ID and version info regularly over DDS
  ➢ Emits a backup broadcast beacon on all available interfaces
  ➢ Emits a heartbeat to the topside to show connectivity

❖ Client discovers vehicles by listening on Beacon topic
  ➢ Client uses wildcard partition in this instance

❖ When targeting consumer mobile devices:
  ➢ Don't expect multicast discovery to work
    ■ Some devices don’t even have multicast enabled in the kernel
    ■ Some do, but only for IPv6
    ■ Some change how routing is done when LTE is enabled
    ■ It varies across devices, OSes, OS versions, etc.
  ➢ The workaround:
    ■ Broadcast generally seems to work where multicast fails!
      ● Not that it should...
    ■ Emit UDP packets on the broadcast address from the vehicle
    ■ On receiving clients, manually add detected IPs to participant peer list
    ■ Now discovery can continue over unicast!
Build and Deployment Process

GitLab

resin.io

docker
General Update Strategies

❖ Extensible types allow for gradual evolution of messages with a core feature set

❖ Supporting redundant interfaces until obsolescence
  ➢ Monitoring what versions of software are deployed to robots and client devices

❖ Have the vehicle tell the client what it is capable of and let the client instantiate the appropriate interface

❖ Type Assignability QoS properties that relax sequence length and member name strictness:
  ➢ dds.type_consistency.kind
  ➢ dds.type_consistency.ignore_member_names
  ➢ dds.type_consistency.ignore_sequence_bounds
  ➢ Can help to ease minor refactoring without breaking compatibility
Platform Extensibility

- External Payloads
- RTI Connector for Python and Node.js Extensions
- Interfacing with ROS2
External Payloads

- Trident is equipped with several mounting points for external payloads

- Onboard WiFi allows integration of smart payloads
  - Additional cameras
  - Additional processors
  - New sensors
  - Gripper arms
  - And more...

- Can interface with the core software in multiple ways
  - DDS
  - Web-based APIs
  - Simple TCP/UDP interfaces

- IDL, QoS, and generated types
  - Publicly available via GitLab
  - Stored on the vehicle as well
RTI Connector

- RTI Connector is a lightweight library for prototyping and testing DDS applications
  - Available in a number of languages
    - Node.JS
    - Python
    - Lua
    - C (not yet released)
  - Allows users to extend the functionality of their robot by writing plugin apps
  - Potential for creating an online repository for sharing plugins
  - All required files pre-installed on the vehicle!
Interfacing with ROS2

Q: What is ROS?
A: From the ROS2 Wiki:

“The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.”
Interfacing with ROS2

- ROS essentially provides a framework for building robotics applications
  - Many algorithms and applications exist as standalone, plug’n’play nodes
  - Used extensively by researchers, students, and government agencies
    - Historically, great for prototyping
    - Some limitations in the original implementation
  - Comes with a number of tools, including visualization and simulation tools

- Where ROS was built on a custom TCP stack, ROS2 is built to support multiple middleware platforms
  - Supports a few different DDS vendor implementations
  - Tries to abstract most of the middleware details away from the user
    - In most cases, does not expose native DDS types to users
  - Uses a predictable set of conventions with regards to DDS features
    - Variable names
    - Topic names
    - QoS policies
    - Partitioning scheme
Interfacing with ROS2

To borrow a slide from an RTI presentation...

ROS 2 - Built on DDS

Plumbing + ROS usability

less time spent here + means + more time to spent here

Open Source Robotics Foundation

©2015 Real-Time Innovations, Inc.
Currently some issues with integrating native DDS and ROS2 applications

- Potential compatibility issues
  - ROS2 types generated from .msg and .srv files
    - Bound to doing the same for your app
    - Type Consistency relaxation won’t help you on the ROS2 side
  - ROS2 does not currently support
    - Key fields
    - Enums
    - Union Types
    - And some other standard DDS features
  - Must adhere to ROS2 topic and partition naming conventions
  - Framework has a bit of a learning curve
- Many of these are doable from a non-ROS2 app
- Integrating with existing DDS apps from ROS2 is the pain point
Interfacing with ROS2

That said…

❖ ROS2 is very actively in development
  ➢ Open to input and pull requests
  ➢ More underlying DDS functionality is being exposed

❖ Having ROS2 interfaces facilitates academics, researchers, and more to use Trident as a platform
  ➢ For now, we will aim to:
    ■ Follow ROS2 type conventions as much as possible
      ● Units
      ● Names
      ● Sensor type descriptions
    ■ Build a ROS2 bridge to act as an interface to our S/W
  ➢ With some additional features, could eventually run on ROS2
Video Delivery and Recording Subsystem

- Tips for WiFi
- QoS Tuning for Livestreaming
- Camera Discovery
- Camera Settings Interface
- Recording Solutions: Client-side and Vehicle-side Use-Cases
Key Problems to Tackle

❖ Minimizing perceptual impact on users
  ➢ Freezing/Stuttering
    ■ Very uncomfortable after a certain threshold
    ■ If the vehicle is moving at high speeds, potential for bad situations
  ➢ Decoding artifacts
    ■ In H.264, if you miss a P-Frame, image is “corrupted” until refreshed
    ■ Want to avoid as much as possible, while not freezing the stream!

❖ Striking a balance between latency and reliability
  ➢ Several tradeoffs to be made
    ■ Minimize redundant data transfers (retries, ACKNACKs, heartbeats, etc.)
    ■ Minimize average time until missing frames are detected so they can be repaired

❖ Minimize the negative impacts of transmission over WiFi
H.264 Decoding Artifacts
Tips for WiFi

❖ Have a general understanding of the 802.11 protocols
  ➢ Management frames, control frames, data frames
  ➢ Airtime fairness and scheduling policies
    ■ Very serial in nature
    ■ Very often, significant amount of time spent not transmitting data

❖ Understand the capabilities of your router, both at the hardware level and software level
  ➢ Routers/radios have a QoS scheme of their own!
  ➢ Many WiFi chips implement different features that can help or harm your application
  ➢ Driver support and maturity is important
    ■ LEDE is an example of great improvements in the Atheros driver stack

❖ Disable support for older protocols, like 802.11b, if possible
  ➢ You can sometimes find your AP limited to the speed of the slowest station
  ➢ Almost all modern devices support 802.11n or better

❖ Increasing short/long retries can help with latency/throughput when using DDS
  ➢ At the 802.11 level, router is generally more efficient at retrying a packet than software
  ➢ You also get the opportunity of retrying UDP fragments, instead of full packets
Tips for WiFi

❖ Understand MIMO configurations
   ➢ 1x2, 2x2, 2x3, etc...
   ➢ Difference between diversity and multiplexing
     ■ Spatial diversity reduces impact of fading and interference - Increased reliability!
     ■ Spatial multiplexing parallelizes data transfer - Increased throughput!

Example:

Transmitter

Returner

2x2 MIMO configured for spatial diversity

❖ Four different paths that data can take
❖ Higher chance that data will make it
❖ Potentially less susceptible to:
   ➢ Device orientation
   ➢ Physical obstacles
   ➢ Outside interference
Writer QoS Tuning

❖ Turn off positive acknowledgement
  ➢ Extra ACK traffic on every frame increases overhead
    ■ disable_positive_acks = true

❖ Piggyback heartbeats on every frame
  ➢ Efficient way of making sure reader gets a chance to catch up
    ■ min/max_send_window_size == heartbeats_per_max_samples

❖ Don’t wait to respond to NACKs
  ➢ Helps reduce repair latency
    ■ min/max_nack_response_delay = 0

❖ Choose a reasonable heartbeat period
  ➢ Too low can cause unnecessary overhead
  ➢ Too high can increase latency of repair
  ➢ We use around half a frame period
    ■ heartbeat_period = 20ms
**Writer QoS Tuning**

- Suppress redundant NACKs for an adequate amount of time
  - Prevents you from saturating link with retries when data already in flight
  - We found that around once per frame works well
    - `nack_suppression_duration = 34ms`

- Find a balance between delaying the stream and dropping a frame
  - Freezes of ~200ms or longer start to become very uncomfortable
  - Streams with faster IDR refresh rates can drive this down even lower
    - `disable_positive_acks_min/max_sample_keep_duration = 210ms (~6 frames)`

*GOP = 90*

Up to 210ms frozen while retrying...

... I P P ...

... 84 frames later ...

P P I ...

I-Frame resets the image

If frame is lost, stream will have artifacts for almost 3 seconds!
Writer QoS Tuning

- Suppress redundant NACKs for an adequate amount of time
  - Prevents you from saturating link with retries when data already in flight
  - We found that around once per frame works well
    - `nack_suppression_duration` = 34ms

- Find a balance between delaying the stream and dropping a frame
  - Freezes of ~200ms or longer start to become very uncomfortable
  - Streams with faster IDR refresh rates can drive this down even lower
    - `disable_positive_acks_min/max_sample_keep_duration` = 210ms (~6 frames)

GOP = 10

- Up to 210ms spent retrying until lost...
- I-Frame resets the image
- If frame is lost, some artifacts until refresh ~60-100ms
Reader QoS Tuning

- Maintain a history length that matches up with your allowable freeze time
  - For 210ms, we keep ~6 frames
    - depth = min/max_sample_keep_duration / frame period
  - Can also use lifespan policy to control this aspect

- Don't wait to respond to heartbeats
  - Reduces repair latency
    - min/max_heartbeat_response_delay = 0ms

- Send a periodic NACK that is faster than your video framerate, if you can
  - Helps reduce repair latency
  - We send a nack every 5ms, so about 6 NACKs per frame
    - nack_period = 5ms

- Disable reader filtering of redundant NACK emissions
  - Let the datawriter do the work of suppressing NACKs!
  - You want to optimize for alerting the writer that you missed a frame
    - round_trip_time = 0
For more tuning information...

- Check out the RTI Connext User Manual
  - Part 3: Advanced Concepts
    - Reliable Communications
      - Using QosPolicies to Tune the Reliable Protocol
Though our vehicle has only one camera installed, we support the addition of many!
Camera & Channel Topic Hierarchy

**Type: Camera**

- **зоn_cams**
  - **forward**
  - **gopro**

**Camera Instances**

**Type: Channel**

- **rov_cam_forward**
  - **h264_0**
  - **h264_1**
  - **jpeg_0**

- **rov_cam_gopro**
  - **jpeg_0**
Camera Interface

Core Camera Type Descriptions

❖ Types draw heavily on V4L2 API

➢ Camera
  ■ UUID (key)
  ■ Driver Info
  ■ Hardware Bus Info
  ■ Capabilities
  ■ Intrinsic Parameters
  ■ Channels

➢ Channel
  ■ UUID (key)
  ■ Pixel Format
  ■ Capabilities
  ■ Extra

➢ ControlDescriptor
  ■ ID (key)
  ■ Name (human readable)
  ■ Value Type (int, string, etc)
  ■ Unit (percent, degrees, etc)
  ■ Min, Max, Step, Default
  ■ Menu Options

➢ ControlValue
  ■ ID (key)
  ■ Union Type Value
  ■ Trigger Default
  ■ Result
**Camera Interface**

**Core Channel Topics**

<table>
<thead>
<tr>
<th>Type: ControlDescriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>rov_cams_SCAM_$CHAN_ctrl_desc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type: ControlValue Request/Reply Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>rov_cams_SCAM_$CHAN_ctrl_req</td>
</tr>
<tr>
<td>rov_cams_SCAM_$CHAN_ctrl_rep</td>
</tr>
</tbody>
</table>

**Instances | type | min | max | default | menu_options**

- **bitrate**: Int32, 1000 - 10000000, 3000000, N/A
- **brightness**: Int32, -255 - 255, 0, N/A
- **framerate**: Int32, 0 - 30, 30, N/A
- **horizontal_flip**: Boolean, N/A - N/A, false, N/A
- **powerline_freq**: StringMenu, 0 - 2, 0, "None", "50 Hz", "60 Hz"

**Value (Union Type)**

| bitrate | 3000000 |
| brightness | 100 |
| framerate | 30 |
| horizontal_flip | true |
| powerline_freq | 2 |
Camera Interface

Specialized Channel Topics/Types

❖ ImageData
  ➢ Transmission of video or still capture data

❖ ImageCapture
  ➢ Used to request capture of a still frame

❖ VideoRepair
  ➢ Clients can request repairs of missed video frames
  ➢ Frames repaired over lower priority flowcontroller
  ➢ Depth of repair buffer configurable for each channel

❖ VideoStats
  ➢ Dropped frames
  ➢ Average framerate
  ➢ Average bitrate
  ➢ Min/max frame sizes
  ➢ Estimated latency at various stages
Camera Interface

Specialized Channel Topics/Types

❖ Histogram
  ➢ Stream of histogram data for raw capture channels

❖ Metadata
  ➢ Additional metadata for raw channels
    ■ Chroma & Luma info
    ■ Macroblock stats
    ■ Etc

❖ Motion Vectors
  ➢ Raw motion vectors that will be fed to the H.264 encoder
    ■ Potentially useful as tracking/stabilization input

❖ And more…
  ➢ Extend as needed
Two primary modes of recording
- Client-side
- Vehicle-side
- Both happen simultaneously!

Client is able to record whichever channel it has subscribed to as a livestream feed
- Reliability is dictated by DDS reliability protocol
- Sample lost? Gap in the recording!

Client can send RepairRequest to a RepairRequest Reader on any channel
- RepairCache Reader subscribes to primary ImageData topic over shared memory
  - Quickly retrieves frames with matching IDs in repair request
  - This saves them from being overwritten in datawriter cache
- RepairCache Reader can then:
  - Keep data in memory for client to slowly retrieve in the background
  - And/Or...
  - Write repair frames to disk for eventual retrieval by the client
- Vehicle-side recording accomplished by making special request to repair all frames
Recording Architecture

Vehicle
- Livestream Writer
- RepairCache Reader
  - Disk or RAM
- Repair Writer
- RepairRequest Reader

Client Application
- ImageData
- Livestream Reader
  - Render
  - Backup Service
  - Disk
- Repair Reader
- ImageData
- RepairRequest Writer
  - OnLostFrames

Cloud Services
Recording Architecture

❖ Rationale

➢ 95% of the data is already on the client if you record while streaming
  ■ Vehicle destroyed with mission critical data? Not a complete loss.
  ■ Reduced time spent syncing data
    ● 56Mbps to 100Mbps connection slow for large recordings
    ● Instead, you can just sync the frames you missed

➢ Client devices tend to have smaller storage capacity
  ■ Fill up quickly if being used with more than just the ROV

➢ Client application can back your data up to the cloud when you get back to WiFi

➢ Simultaneously recording on vehicle gives future access to multiple client devices
  ■ Multi-user missions
  ■ Loss or inaccessibility of original client device
  ■ Direct cloud sync from vehicle

➢ Since Livestream Writer and Repair Writer are in the same participant...
  ■ Flowcontrol QoS allows for repair in the background while still livestreaming
    ● Minimal interruption
Bridging the Gaps: Where DDS Has Room to Grow

- DDS in Extremely Resource Constrained Environments (XRCE)
- Peer-to-Peer via WebRTC: Can Web DDS go farther?
- Cross-platform DDS API
Towards a global marine intelligence platform
Towards a global marine intelligence platform
Towards a global marine intelligence platform
Where no ROV has gone before… probably
Q&A

Contact:
charles@openrov.com