Developing a MOOS Driver for Marine Broadband Radar

Michael Sacarny, MIT Sea Grant College
Paul Robinette, MIT Sea Grant College
Marine Radar: “Ideal” requirements

- Range: A few meters to \( > 10 \text{ km} \)
- Resolution: Can resolve objects (e.g., pilings) near vessel
- Power requirements: The lower the better, active and standby
- Environmental: Fully marinized to rain, salt spray, snow, heat, cold
- Cost: \(< \$2k\)
- Scan update rate: \(~5 \text{ Hz or better}~\)
- Protocol: Public
- Software/Driver: Open source, Linux/Mac
Navico 4G Broadband radar

✓ Range: 50 m to 48 km
✓ Resolution: Can (sometimes) resolve pilings near vessel
✓ Power: 18 W active, 2 W standby (intensity ⅕ cell phone)
✓ Environmental: Fully marinized
✓ Cost: < $2k
 ~ Update rate: 48 rpm
✓ Bonus: Two channels!
✘ Protocol: Proprietary
✘ Software/Driver: Proprietary, data available on Navico Ethernet
Research: Past efforts


Test setup
Multicast data streams

From Adrian Dabrowski, et. al.
Radar data in packetized form

Radar imagery in polar form [2048 scanlines * 512 px] (a) and packetized form (b).

From Adrian Dabrowski, et. al.
Protocol examination: Wireshark

Data Packets

Radar spoke data
Research: OpenCPN Plugin source code

Radar PI Plugin
Challenges of reverse engineering

- You are at the mercy of the protocol
  - It could changed or be encrypted at any time
- Example: What determines the range of spoke data in the packet?
  - OpenCPN radar plugin discussion:
    - “Why does the radar return such strange units for range in the packet header?”
    - “Why?? Ask Navico. It is as it is. Here we just try to interpret.”
  - Determined empirically, implemented as look-up table
Implementation: MOOS class

Standard AppCastingMOOSApp class

class Radar : public AppCastingMOOSApp
{
friend class Channel; // allow access to protected functions
public:
    Radar();
    ~Radar();

protected: // Standard MOOSApp functions to overload
    bool OnNewMail(MOOSMSG_LIST &NewMail);
    bool Iterate();
    bool OnConnectToServer();
    bool OnStartUp();

protected: // Standard AppCastingMOOSApp function to overload
    bool buildReport();
    string buildChannelReport(int channel);
    void registerVariables();
    void notifySettings();
Implementation: MOOS configuration file

- Full details in Radar_Info.cpp

```
ProcessConfig = iRadar2
{
  AppTick     = 4
  CommsTick   = 4
  AntennaHeight = 2.0
  0.run      = 1
  0.range    = 100
  0.gain     = Auto
  0.noiserej = 3
  1.range    = 250
  1.gain     = 50
}
```
Implementation: Sockets and radar status

- I/O based on (cryptic) BOOST asio::ip::udp::socket classes
- Data passed between background and main thread by mutex-protected data queues
- Small state machine to handle overall radar channel status

```cpp
void Channel::changeStateChannel(int newState)
{
    stateChannelLast = stateChannel;
    switch (newState) {
        case ST_CHANNEL_NO_COMMS:
        case ST_CHANNEL_NO_PULSE:
        case ST_CHANNEL_STANDBY:
        case ST_CHANNEL_SPINNING_UP:
        case ST_CHANNEL_TRANSMITTING:
        case ST_CHANNEL_WINDING_DOWN:
            stateChannel = newState;
            break;
        default:
            parent->reportEvent(chDesc()) + "In cha
            break;
    }
}```
Implementation: MOOS DB variables

- Control changes through variable RADCMD
  - Notify ("RADCMD", "0.gain = auto")
- Status reports are available through variable RADRPT
  - RADRPT = 0.Range = 750
- Data packets available through variable RADRAW[0,1]
  - RADRAW0 = [spoke data]
REx4 and Philos installations
Example: Yacht Club
Example: Museum of Science
Future work

- Better post-processing of data to reduce “blobbiness” and identify targets
- Algorithmic selection of optimal gain, range, and other control settings
- Exploit multiple radar channels to provide better overall coverage
- Fusion with other data sources
- Public release!
Acknowledgements

Many thanks to our corporate sponsor Brunswick Corporation and Mercury Marine!