Introduction to Robotic Technology Kernel (RTK)

TARDEC Ground Vehicle Robotics
Presentation Overview

• RTK Overview
• Architecture
• Vehicle Configuration
• Development
• Useful Tools
RTK Overview
Autonomy Kit/Drive-by-Wire Kit Concept

Operator Control Unit

Autonomy Kit

Drive-By-Wire Kit

UNCLASSIFIED: Distribution Statement A. Approved for public release; distribution is unlimited.
The Robotic Technology Kernel (RTK) is a Robot Operating System (ROS)-based modular autonomy software library for S&T development that provides a set of common robotic capabilities across a variety of platforms and efforts.

Cross-Effort
Efforts deliver new capabilities to the RTK, which in turn is leveraged for new efforts.

Cross-Controller
Any RTK-enabled platform can be controlled by any RTK compatible controller or WMI.

Cross-Platform
New RTK capabilities are enjoyed by all RTK-enabled platforms.

Robotic Technology Kernel (RTK)
The primary RTK software library is maintained in the RTK Master Branch. The Master Branch contains nearly all available RTK capabilities with the exception of specialized mission or project specific functionality.

RTK Core is a documented and distributable version of RTK. RTK Core is focused on providing reliable, general purpose autonomy software and is released on a periodic basis.

The primary RTK software library is maintained in the RTK Master Branch. The Master Branch contains nearly all available RTK capabilities with the exception of specialized mission or project specific functionality.

RTK Core is a documented and distributable version of RTK. RTK Core is focused on providing reliable, general purpose autonomy software and is released on a periodic basis.

Specialized instantiations of RTK are maintained in project branches, which originate with the RTK Master Branch or an existing project branch.

Capabilities and functionality that have broad application to robotic autonomy in general get integrated back into the RTK master branch.

During the course of a project, new capabilities may be developed and existing capabilities may be changed to better meet project goals.
### RTK Past and Present

<table>
<thead>
<tr>
<th>FY13</th>
<th>FY14</th>
<th>FY15</th>
<th>FY16</th>
<th>FY17</th>
<th>FY18</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSAT</td>
<td></td>
<td>MUER</td>
<td></td>
<td></td>
<td>Wingman JCTD</td>
</tr>
<tr>
<td>Wingman</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TORVICE</td>
<td>AMAS MTV / AGR-TR / AGR Behaviors</td>
<td>CAAR</td>
<td>RTK Core</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DSAT** – Dismounted Soldier Autonomy Tools  
**MUER** – Multi UGV Extended Range  
**TORVICE** – Trusted Operation of Robotic Vehicles in a Contested Environment  
**AMAS** – Autonomous Mobility Applique Kit  
**AGR** – Autonomous Ground Resupply  
**CAAR** – Coalition Assured Autonomous Resupply

In addition to the development efforts listed here, RTK has been and continues to be used on several other projects to provide autonomous capability.
Architecture
This subsystem contains the sensor drivers and algorithms that detect and interpret things of interest from the environment.

- Stereo Cameras
- LIDARs
- Ultra Wideband Radios (UWBs)
- RADAR

- Output:
  - Disparity
  - Material Classification
  - Ground Point Clouds
  - Non-Ground Point Clouds
Localization

This subsystem fuses data from multiple sensors to provide both relative and absolute pose of the vehicle.

- **GNSS**
  - GPS
  - DRTK
  - DGPS

- **Wheel Speed Encoders**

- **IMU**

- **Gyro**

- **RANGER**

- **Visual Odometry**

- **Output:**
  - far_field -> near_field -> vehicle TF tree
    - map -> odom -> base_link in standard ROS terms
  - far field and near field odometry
World Model

This subsystem is used for storing and fusing data from multiple sources. Services are provided to request data as a costmap or list of dynamic objects and zones.

• Input:
  – Plugin-based sensor fusion from Perception & Localization
  – “No go” zones
  – Known Obstacles

• Internal operations:
  – Octree-based sparse voxel map
  – Persistent Map

• Output:
  – Voxels
  – 2D Costmaps
This subsystem configures and monitors behavior modules.

- **Input:**
  - Mode commands

- **Output:**
  - Path planner configuration
Navigation

This subsystem generates paths and speed/steering commands to guide the vehicle to its destination.

• Input:
  – Filtered 2D Costmaps
  – Vehicle Odometry

• Internal operations:
  – Behavior switch
  – Path Planning
    • A*
    • Maverick
    • Vaquero
    • Vaquerito

• Output:
  – Speed & Curvature Setpoint
This subsystem monitors overall health and ensures commands are safe to perform.

• Input:
  – Direct teleoperation
  – Mode commands
  – Camera control and configuration
  – Speed and steering
  – Vehicle status

• Output:
  – Mode confirmation
  – Verified speed and steering
Motion Execution

This subsystem handles primitive control and status between the autonomy kit and the drive-by-wire kit.

• **Input:**
  – Speed & Curvature Setpoint
  – Vehicle Odometry

• **Output:**
  – Drive-by-Wire Commands
Other Subsystems

• Diagnostics
  – This subsystem provides a way to monitor the health of hardware in the system.

• IOP Bridge
  – This subsystem is the interface between the autonomy kit and the operator control unit.

• CAN-A-Kit Bridge
  – This subsystem handles traffic to and from the CAN bus.

• Payload System
  – This subsystem handles control, configuration, and status of pan/tilt, camera, and remote weapon systems that can be controlled by the operator.
Vehicle Configuration
Sensors

- GNSS*
  - GPS
  - RTK
  - DGPS
- IMU*
- LIDAR*
- Wheel Speed Encoders**
- Gyro**
- Stereo Cameras
- Teleop Cameras
- Data Radios
- UWB Radios
- RADAR

* Required
** Not required, but a really good idea
Computers

• Main
  – ROS Master
  – Navigation
  – World Model
  – Motion Execution

• Localization
  – Hardware Drivers
    • GPS
    • IMU
    • Wheel Speed
  – State Estimation

• Vision
  – Camera Drivers
  – Image Rectification
  – Disparity Segmentation
  – Material Classification

• LIDAR
  – Velodyne Drivers
  – Ground Segmentation
Development
Workflow

• Create a ROS workspace
  – Clone repositories
  – Configure the workspace
  – Install dependencies
  – Build it
• Make changes
  – Edit code
  – Deploy to a vehicle
  – Test the changes
  – Push them to a new branch
  – Create a Merge Request to request a code review
Deployment (Traditional)

• “deploy” script copies compiled binaries to /opt/rtk/indigo (or kinetic)
• Ubuntu 14.04 / ROS Indigo:
  – Upstart script in /etc/init/dsat.conf
  – sudo service dsat start/stop/restart
• Ubuntu 16.04 / ROS Kinetic:
  – Systemd unit in /etc/system/system/dsat.service
  – sudo systemctl start/stop/restart dsat
Developer Resources

- **DI2E**: Widely-Available Release Repository
  - RTK Core
  - WMI Core
- **Dismount-Git**: Primary Development Repository
  - RTK Master Branch
  - RTK Project Branches
- **Developer Handbook**: Environment Setup, Development Workflow, Style Guide, etc.
- **RTK User Guide**
  - Architecture Diagrams, Flow Charts, Use Cases
- **RTK Architecture Guide**
  - Low-level ROS package documentation
  - ROS node APIs
Useful Tools
Calibration

• Stereo Camera Calibration
  – Create calibration boards
  – Collect data
  – Perform calibration
  – Apply calibration files

• LIDAR Alignment
  – Necessary if using more than one LIDAR
  – Launch the calibration system
  – Run rviz to monitor progress
  – Use calibration system to manually align LIDARs
  – Save calibration values into platform launch file
Playback Mode

- `roslaunch mrzr_8803 mrzr_8803.launch playback:=true`
- Runs all nodes on the local computer
- Disables hardware drivers
- Useful for playing back bag files to debug behavior
- `rosbag play -clock -r 0.5 rtk_raw_test_track_playback*.bag \ local_xy_origin:=local_xy_old`
swri_console

- Like rosconsole but better
• Top-down ROS visualization tool, similar to rviz
Bag Database

- Searchable, inspectable database of bag metadata
• Displays diagnostic messages
• Useful on a live vehicle to tell if all the hardware is working
Questions?