WEBINAR SERIES ON ADVANCED MOBILITY
WEBINAR SERIES ON ADVANCED MOBILITY

Multi-Vehicle Mission Scenarios

01 April 2024

Dr. Marc Compere, Associate Professor
(comperem@erau.edu, https://faculty.erau.edu/Marc.Compere)
Department of Mechanical Engineering
Embry-Riddle Aeronautical University, Daytona Beach FL, USA
Agenda

1. Brief Intro and Background
2. The Mobility Virtual Environment
3. Multi-Vehicle Testing: Virtual and Real
   1. Real vehicle tests, outdoors, with references
   2. Simulation for the remainder of this presentation
4. MoVE architecture:
   1. High-level Diagram
   2. Individual Vehicle Models
   3. Priority-based Behavior Table
   4. Vehicle-to-Vehicle Communication Model (v2v)
   5. Mission Sequencer Table
5. Mission Rehearsal Example
6. The complete Mission Simulation Architecture
7. Documentation: Code, Docs, References
Motivation
It’s happening!
https://arxiv.org/abs/2402.18062
The Need

The UAS research community needs a way to rehearse multi-vehicle scenarios along a realism spectrum from simulation-only to all real vehicles with real people.

Ideally, the community needs:
• Common tools for easily sharing scenarios and data
• Open-source software available to many*
  note: *the US military has good open-source policy
• Simple software that Just Works with standard computers (no hardware or high-end graphics dependencies)
• A way to leverage common devices: iPhones, Androids, Raspberry Pi / Arduino
• Methods that capture wireless networks, including cellular networks
Introduce **MoVE**
the
**Mobility Virtual Environment**
**MoVE: Mobility Virtual Environment**

MoVE is a network-centric framework for testing real vehicles, virtual vehicles, and pedestrians in the same coordinate system with a common timestamp.

Real and Virtual pedestrians

Real Vehicle, Real Operators

Virtual vehicles, Repeatable traffic

Repeatable challenge cases
MoVE Vehicle Motion

There are three distinct ways of using MoVE:

1. **Simulation Only**

2. **Real Vehicles and Real Pedestrians**
   - Stream GPS locations over wireless network:
     - iPhone SensorLog app, Android HyperIMU app
   - Motion in the real world is conveyed to the virtual environment

3. **Mixed Real and Virtual**:
   - Virtual vehicles can avoid real people
   - Real vehicles can see repeatable virtual challenges
     (with AR headset)

Today’s presentation!

References slide has [1-6], but mainly

Field Campaign with Real Vehicles, Arizona, 2021

- MoVE was critical for gathering multiple vehicle locations and sensor data across 2 weeks of flights!
MoVE Multi-Vehicle Simulations
MoVE Software Architecture

How does a MoVE experiment work?

1. Read config file, Launch processes

2. Issue runState commands:
   Ready, Set, Go!, Pause, Stop

3. All vehicles report position and health status to Core

4. Core aggregates all positions into State and logs with timestamps

MoVE Core

State

vehicle 1
vehicle 2
⋮
vehicle N

udp/ip

live 2D map updates
record
playback

Database

Live 2D Map
A Simple Kinematic Vehicle Model

MoVE’s vehicle models have a characteristic length, $L$, but otherwise are similar to:

- Dubins Car [1]
- Dubins Airplane [2]

Kinematic equations of motion in the body-fixed $xy$ frame are:

$$ v_g^{xy} = \begin{bmatrix} v_x \\ b\dot{\psi} \end{bmatrix} \quad \dot{\psi} = \left( \frac{v_x}{L} \right) \delta_{steer} $$

Transform body-fixed velocities in $xy$ frame to the inertial $XY$ frame with:

$$ v_g^{XY} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \cdot v_g^{xy} $$

Then integrate $XY$ velocities for $XY$ positions:

$$ \begin{bmatrix} X_g \\ Y_g \end{bmatrix} = \int v_g^{XY} dt \quad \psi = \int \dot{\psi} dt $$

Rodney Brooks in the late 1980’s introduced the concept of fast, reactive behaviors based on sensor inputs [1].

Behaviors are separate threads within a vehicle process

Behaviors with highest priority determine vehicle motion

<table>
<thead>
<tr>
<th>idx</th>
<th>behavior</th>
<th>priority</th>
<th>enable?</th>
<th>steer cmd</th>
<th>throttle</th>
<th>pitch cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>circleInPlace</td>
<td>3</td>
<td>(0/1)</td>
<td>const</td>
<td>const</td>
<td>const</td>
</tr>
<tr>
<td>2</td>
<td>multiRotorHover</td>
<td>3</td>
<td>(0/1)</td>
<td>hold</td>
<td>holdElev</td>
<td>hold</td>
</tr>
<tr>
<td>3</td>
<td>followRoute</td>
<td>5</td>
<td>(0/1)</td>
<td>$\delta = f(\psi_{err})$</td>
<td>$v_i = f(s_i)$</td>
<td>$\alpha = f(\theta_{err})$</td>
</tr>
<tr>
<td>4</td>
<td>gotoGate</td>
<td>6</td>
<td>(0/1)</td>
<td>$\delta = f(\psi_{err})$</td>
<td>$v_i = f(g_i)$</td>
<td>$\alpha = f(\theta_{err})$</td>
</tr>
<tr>
<td>5</td>
<td>gotoPoint</td>
<td>7</td>
<td>(0/1)</td>
<td>$\delta = f(\psi_{err})$</td>
<td>$v_i = f(p_i)$</td>
<td>$\alpha = f(\theta_{err})$</td>
</tr>
<tr>
<td>6</td>
<td>stayInBounds</td>
<td>20</td>
<td>(0/1)</td>
<td>turnAround</td>
<td>+10%</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>avoidBySteer</td>
<td>15</td>
<td>(0/1)</td>
<td>turnAway</td>
<td>const</td>
<td>turnAway</td>
</tr>
<tr>
<td>8</td>
<td>avoidBySpeed</td>
<td>17</td>
<td>(0/1)</td>
<td>const</td>
<td>avoid</td>
<td>const</td>
</tr>
<tr>
<td>9</td>
<td>avoidByElevation</td>
<td>19</td>
<td>(0/1)</td>
<td>const</td>
<td>const spd</td>
<td>avoid</td>
</tr>
</tbody>
</table>

Vehicle inputs, $u_{beh}$

Vehicle models with behaviors, mobility dynamics, and RK4 integrator are advanced in soft-real-time.
Behaviors enabled in default.cfg

`{ 'wander': 1, 'periodicTurn': 2, 'stayInBounds': 10, 'avoidBySteer': 15 }`
v2v
Vehicle-To-Vehicle Communication

• A very simple broadcast model of v2v communications is implemented.

• udp/ip multicast is a simple way to model a broadcast network among all vehicles

• New threads:
  – v2vUpdateToNetwork()
  – v2vUpdateFromNetwork()

• ¡Voila! v2v communications!

<table>
<thead>
<tr>
<th>vid</th>
<th>name</th>
<th>lat</th>
<th>lon</th>
<th>elev</th>
<th>M.S.</th>
<th>% complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>joe</td>
<td>0</td>
<td>1</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>jim</td>
<td>10</td>
<td>5</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>bill</td>
<td>20</td>
<td>7</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each vehicle reports and receives v2v messages and builds its own table of what every other vehicle (and pedestrian) is doing.
a Mission Sequencer
Multi-vehicle missions are more than behaviors and communications.

Missions include time and space-dependent conditions across multiple vehicles, with goals for all vehicles.

A mission for a single vehicle needs to be described as a sequence of steps that can be completed – to complete the mission.

“Wait here 2 minutes, then go to the corner.”

When I arrive at the corner, we both will drive back to base.”
The Multi-Vehicle Mission Language (MML)

Mission Commands for Both:

- vid100:
  - takeOff()
  - moveTo( A1 )
  - waitUntil( vid101 @ A2 )
  - land()

- vid101:
  - waitUntil( vid100 @ A1 )
  - takeOff()
  - moveTo( A2 )
  - land()
a Mission example:
missionDefault.cfg
# a mission table is constructed from rows of commands, for a particular vehicle identifier (vid)
# labels prior to the ':' have no influence but must be unique; they do not need to be ordered
# each vid's set of commands are read (in any order) and sorted by missionState
# required: 'vid', 'missionState', 'action'; all others may or may not contain action-specific keys

```yaml
# Newton

mission_cmd00: {'vid': 100, 'missionState': 2, 'action': 'waitTilElapsed', 'eTimeThresh': 4.0} # (s) wait this many elapsed seconds within the mission sequencer

mission_cmd01: {'vid': 100, 'missionState': 10, 'action': 'goToPoint', 'point': 'first', 'speed': 3, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd02: {'vid': 100, 'missionState': 15, 'action': 'waitTilElapsed', 'eTimeThresh': 4.0} # (s) wait this many elapsed seconds within the mission sequencer

mission_cmd03: {'vid': 100, 'missionState': 20, 'action': 'goToPoint', 'point': 'second', 'speed': 3, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd04: {'vid': 100, 'missionState': 25, 'action': 'goToPoint', 'point': 'third', 'speed': 3, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd05: {'vid': 100, 'missionState': 30, 'action': 'goToPoint', 'point': 'home', 'speed': 3, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd06: {'vid': 100, 'missionState': 35, 'action': 'waitTilElapsed', 'eTimeThresh': 4.0} # (s) wait this many elapsed seconds within the mission sequencer

mission_cmd07: {'vid': 100, 'missionState': 40, 'action': 'goToPoint', 'point': 'newMissionState', 'newMissionState': 2, 'maxIterations': 200} # 'maxIterations' is max number of goTo return

mission_cmd08: {'vid': 100, 'missionState': 45, 'action': 'goToMissionState', 'newMissionState': 100, 'action': 'waitOnVehProgress', 'otherVeh': 100, 'progThreshold': 11.0} # otherVeh's progress must be -> threshold

# MarieCurie

mission_cmd10: {'vid': 101, 'missionState': 1, 'action': 'goToPoint', 'point': 'first', 'speed': 5, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd11: {'vid': 101, 'missionState': 10, 'action': 'goToPoint', 'point': 'second', 'speed': 5, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd12: {'vid': 101, 'missionState': 20, 'action': 'waitOnVehProgress', 'otherVeh': 100, 'progThreshold': 20.0} # otherVeh's progress must be -> threshold

mission_cmd13: {'vid': 101, 'missionState': 25, 'action': 'goToPoint', 'point': 'third', 'speed': 5, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd14: {'vid': 101, 'missionState': 30, 'action': 'goToPoint', 'point': 'home', 'speed': 5, 'riseRate': 3, 'rThresh': 3.0, 'arrivalType': 'stop'}

mission_cmd15: {'vid': 101, 'missionState': 35, 'action': 'waitOnVehProgress', 'otherVeh': 100, 'progThreshold': 31.0} # otherVeh's progress must be -> threshold

mission_cmd16: {'vid': 101, 'missionState': 40, 'action': 'goToPoint', 'point': 'newMissionState', 'newMissionState': 100, 'action': 'waitOnVehProgress', 'otherVeh': 100, 'progThreshold': 11.0} # otherVeh's progress must be -> threshold
```

Newton
vid=100

MarieCurie
vid=101
Behaviors enabled in `missionDefault.cfg`

```json
{
  'multiRotorHover': 1,
  'goToPoint': 5,
  'stayInBounds': 12,
  'avoidBySteer': 10
}
```

MoVE Video of: default.cfg

[https://youtu.be/zNYqOKASJCc](https://youtu.be/zNYqOKASJCc)
Mission Sequencer Functionality

• What just happened? How did that work?

• Each vehicle’s Mission Sequence specifies a **mission action**, with clear criteria on when it is complete.

• Each mission action has a metric (time or distance) that can be tracked, from 0% to 100% complete.

• When the current mission command is 100% complete, **missionState** advances to the next mission command.

• **A complete mission is**: a sequence of **mission actions**, going from point-to-point, and perhaps waiting a duration, or waiting on another vehicle to arrive somewhere.

<table>
<thead>
<tr>
<th>Mission Actions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>goToPoint (pt)</td>
</tr>
<tr>
<td>waitElapsed (dt)</td>
</tr>
<tr>
<td>waitOnVehProgress (other, thresh)</td>
</tr>
<tr>
<td>goToMissionState (newMS)</td>
</tr>
</tbody>
</table>
The Mission
Architecture
**move**

**Built-in Vehicle Model**

**missionCmd changes behavior priorities**

<table>
<thead>
<tr>
<th>i</th>
<th>action</th>
<th>i+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gotoPoint( Pt₁ )</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>waitElapsed( 30s )</td>
<td>11</td>
</tr>
<tr>
<td>20</td>
<td>waitOnVehProgress( otherVeh, threshold )</td>
<td>21</td>
</tr>
<tr>
<td>30</td>
<td>gotoMissionState( 100 )</td>
<td>31</td>
</tr>
<tr>
<td>100</td>
<td>gotoPoint( landPt₁ )</td>
<td>101</td>
</tr>
</tbody>
</table>

---

**v2v table**

<table>
<thead>
<tr>
<th>vid</th>
<th>name</th>
<th>lat</th>
<th>lon</th>
<th>elev</th>
<th>M.S.</th>
<th>% complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>joe</td>
<td>0</td>
<td>1</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>jim</td>
<td>10</td>
<td>5</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>bill</td>
<td>20</td>
<td>7</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**behavior table**

controls vehicle
Code
Docs
References
Documentation site:
https://comperem.gitlab.io/move/

Code:
https://gitlab.com/comperem/move
Conclusion

• MoVE started as an open-source project in 2018

• MoVE provides multi-vehicle, multi-domain simulation, with a v2v model, behavior models, and a Multi-Vehicle Mission Sequencer.

• MoVE also accepts ADS-B, Wifi, Xbee, and other networks for collecting live telemetry from multiple (real) vehicles in field tests.

• Questions and inquiries: Marc Compere, comperem@erau.edu !!!
MoVE Papers
Article References


IEEE VTS Acknowledgement

Dr. Compere would like to acknowledge the IEEE Vehicular Technology Society (https://vtsociety.org/) for sponsoring the Webinar Series on Advanced Air Mobility!

(https://vtsociety.org/post/announcement/webinar-series-advanced-air-mobility)
Join IEEE VTS at www.vtsociety.org

Follow IEEE VTS on social media

- Website: www.vtsociety.org
- Facebook: facebook.com/IEEEVTS
- Twitter: @IEEE_VTS
- LinkedIn: www.linkedin.com/company/ieee-vehicular-technology-society
Real Vehicles and Real Network Telemetry

1. Simulated MoVE vehicles
2. GPS updates over cellular network
3. Xbee 2.4Ghz mesh network
4. ADS-B 1090MHz
5. RemoteID, Bluetooth
6. Lora Mesh with CDP
7. Cloud upload

MoVE Core

State

Computer processes

Rx from real vehicles

Computer processes

Tables and Maps in Browsers

Scenario control dashboard

Map display

Logging

Cloud

ADS-B

TBD

Telemetry 1

Telemetry 2

Telemetry 3

... 

Telemetry N

Veh proc 1

Veh proc 2

Veh proc 3

... 

Veh proc N

UDP/IP

Rx from real vehicles