# COSC-6590/GSCS-6390

## Game Theory

# Game Theoretic Potential Field for Autonomous Water Surface Vehicle Navigation Using Weather Forecasts

Evan Krell and Luis Rodolfo Garcia Carrillo

School of Engineering and Computing Sciences
Texas A&M University - Corpus Christi, USA

# Table of contents

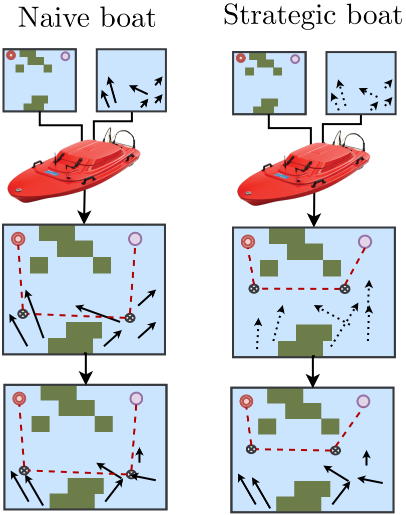
# Problem Description

# Scenario: Autonomous Surface Vehicles' Navigation


**Autonomous surface vehicles**

- Complex coastal navigation
- Dynamic, uncertain weather
- Energy-efficient planning desired

# Addressing Environment Uncertainty




**Naive boat**

- Longer path to ride currents

What if weather is worse than expected?

- Boat regrets its plan

**Strategic boat**

- Path avoids worst case

What if weather better than expected?

- Boat does not regret plan

# Contributions of this Research

**Robust autonomous water surface vehicle navigation**

- Game theory: applied to dynamic programming motion planning for strategic planning in uncertain environments
- Dynamic programming: each iteration yields a feasible plan, with additional iterations optimizing the plan

**Strategy makes use of real data**

- Large marine region and online water forecasts

**Outcomes**

- Plan optimal paths that avoid worst-case weather
- High complexity is offset since effective plans can be generated using far fewer iterations than theoretical bound

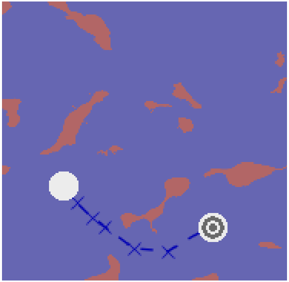
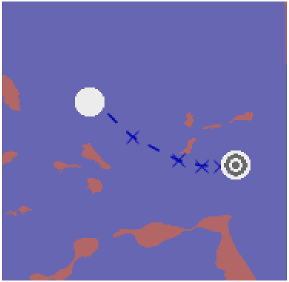
**Software Package Released for Robust Vehicle Navigation**

- `https://github.com/ekrell/fujin`

# Game Theory Motion Planning

# Conventional Path Planning

Typically, a single path is generated from a pre-defined start

Example of paths found using Particle Swarm Optimization

**Common Problem:**

- if the robot gets off-course, it must generate new solution

# Our Approach: Vector Field Motion Plan

**Motion plan to reach goal G**

| | | | | | | |
|---|---|---|---|---|---|---|
| ↓ | ↓ | ↙ | ↙ | X | X | ↓ |
| ↓ | ↓ | ↙ | ↙ | X | X | ↓ |
| ↓ | ↓ | ↙ | ↙ | X | X | ↓ |
| ↓ | ↓ | ↙ | ↙ | ↘ | ↓ | ↙ |
| ↘ | ↓ | X | X | X | ↓ | ↙ |
| ↘ | ↘ | X | X | X | ↙ | ↙ |
| → | → | → | G | ← | ← | ← |
| ↗ | ↗ | ↗ | ↑ | ↖ | ↖ | ↖ |

- Here, plan for entire region
- Path from any reachable cell
- Compare: artificial potential field

**Dynamic Programming**

- generate optimal motion plan in a bottom-up fashion, starting from the goal

# Dynamic Programming

**Cost Map**

| | | | | | |
|---|---|---|---|---|---|
| 4 | 4 | 4 | X | X | X |
| 4 | 3 | 3 | X | X | 3 |
| 4 | 3 | 2 | 2 | X | 2 |
| 4 | X | 2 | 1 | 1 | 1 |
| 5 | X | X | 1 | G | 1 |
| 4 | 3 | 2 | 1 | 1 | 1 |

**Path via Cost Lookup**

| | | | | | |
|---|---|---|---|---|---|
| 4 | 4 | 4 | X | X | X |
| 4 | 3 | 3 | X | X | 3 |
| 4 | 3 | 2 | 2 | X | 2 |
| 4 | X | 2 | 1 | 1 | 1 |
| 5 | X | X | 1 | G | 1 |
| 4 | 3 | 2 | 1 | 1 | 1 |

- Each cell has cost to reach **G**
- Each cost depends on other, previously solved, cells
- Resulting map can generate paths

# Environment Uncertainty

**Sources of uncertainty for the scenario at hand**

- Obtained from model error (*w.r.t* monitoring stations)
- Learned by robot by several missions

**Predicted water force**

**Error margin of water force**

**Forecast uncertainty within estimated error range**

- Otherwise, weather unlimited → predictions useless

# Game Theory–inspired robust actions

Each cell's cost the solution of a 0-sum, 2-player game

- Uncertainty: weather has discrete range of actions
- The range based on the prediction's error margin
- Higher action resolution → higher game complexity

| | | | | | |
|---|---|---|---|---|---|
| Halt | INF | INF | INF | INF | INF |
| ↑ | 146 | 146 | 147 | 148 | 149 |
| ↓ | 91 | 90 | 89 | 89 | 88 |
| ← | 149 | 148 | 147 | 147 | 146 |
| → | 132 | 132 | 133 | 134 | 135 |

Game between 4-way robot and weather, 5 error choices

- Row-player minimizes: **blue**
- Column-player maximizes: **red**
- Mini-max solution: **purple**

# Algorithms & Problem Formulation

# Algorithm Overview: Single Iteration

Bottom-up DP: solves cells to move, starting from goal

- Cell cost: work of action at cell + all other cells to goal

Initial map
empty cells: max cost

| | | |
|---|---|---|
| | G | |
| | | |
| X | X | X |
| X | X | X |
| X | X | |
| | | |
| | | |

Bottom-up DP
after one loop

| | | |
|---|---|---|
| $1^{\rightarrow}$ | G | $1^{\leftarrow}$ |
| $1^{\nearrow}$ | $1^{\uparrow}$ | $1^{\nwarrow}$ |
| X | X | X |
| X | X | X |
| X | X | |
| | | |
| | | |

Bottom-up DP
completed iteration

| | | | | |
|---|---|---|---|---|
| $2^{\rightarrow}$ | $1^{\rightarrow}$ | G | $1^{\leftarrow}$ | $2^{\leftarrow}$ |
| $2^{\nearrow}$ | $1^{\nearrow}$ | $1^{\uparrow}$ | $1^{\nwarrow}$ | $2^{\nwarrow}$ |
| $3^{\nearrow}$ | X | X | X | $3^{\nwarrow}$ |
| $4^{\uparrow}$ | X | X | X | $4^{\uparrow}$ |
| $5^{\uparrow}$ | X | X | $5^{\nearrow}$ | $5^{\uparrow}$ |
| $6^{\uparrow}$ | $6^{\nwarrow}$ | $6^{\nearrow}$ | $6^{\nearrow}$ | $6^{\uparrow}$ |
| $7^{\uparrow}$ | $7^{\nwarrow}$ | $7^{\nearrow}$ | $7^{\nearrow}$ | $7^{\uparrow}$ |

Problem: each solution based on incomplete information!

- Solution: iterative dynamic programming

# Algorithm Overview: Multiple Iterations

Initial map
strong currents

| | | | |
|---|---|---|---|
| | | G | |
| | | | |
| $\downarrow$ | X | X | X |
| $\downarrow$ | X | X | X |
| $\downarrow$ | X | X | |
| | | | |
| | | | |

First iteration
high costs near current

| | | | | |
|---|---|---|---|---|
| $2^{\rightarrow}$ | $1^{\rightarrow}$ | G | $1^{\leftarrow}$ | $2^{\leftarrow}$ |
| $2^{\nearrow}$ | $1^{\nearrow}$ | $1^{\uparrow}$ | $1^{\nwarrow}$ | $2^{\nwarrow}$ |
| $10^{\nearrow}$ | X | X | X | $3^{\nwarrow}$ |
| $18^{\uparrow}$ | X | X | X | $4^{\uparrow}$ |
| $26^{\uparrow}$ | X | X | $5^{\nearrow}$ | $5^{\uparrow}$ |
| $27^{\uparrow}$ | $28^{\nwarrow}$ | $6^{\nearrow}$ | $6^{\nearrow}$ | $6^{\uparrow}$ |
| $28^{\uparrow}$ | $7^{\nearrow}$ | $7^{\nearrow}$ | $7^{\nearrow}$ | $7^{\uparrow}$ |

Second iteration
three actions changed

| | | | | |
|---|---|---|---|---|
| $2^{\rightarrow}$ | $1^{\rightarrow}$ | G | $1^{\leftarrow}$ | $2^{\leftarrow}$ |
| $2^{\nearrow}$ | $1^{\nearrow}$ | $1^{\uparrow}$ | $1^{\nwarrow}$ | $2^{\nwarrow}$ |
| $10^{\nearrow}$ | X | X | X | $3^{\nwarrow}$ |
| $18^{\uparrow}$ | X | X | X | $4^{\uparrow}$ |
| $26^{\uparrow}$ | X | X | $5^{\nearrow}$ | $5^{\uparrow}$ |
| $8^{\searrow}$ | $7^{\rightarrow}$ | $6^{\nearrow}$ | $6^{\nearrow}$ | $6^{\uparrow}$ |
| $8^{\nearrow}$ | $7^{\nearrow}$ | $7^{\nearrow}$ | $7^{\nearrow}$ | $7^{\uparrow}$ |

Each iteration's choices dependent on neighborhood

- Center: chooses high-cost since neighbors not solved
- Right: cells choose lower-cost neighbors next time
- Several iterations for lower-cost strategies to propagate

# Problem Formulation: Environment

Occupancy Grid $\mathcal{R} := M \times N$ matrix where

- Value of 1 indicates occupied
- Value of 0 indicates free

J := number of force vectors affecting $R$

- $u$ components: force grids $F^u := \{F_1^u, F_2^u, \ldots, F_J^u\}$
  - each component is an $M \times N$ matrix of weather force
- $v$ components: force grids $F^v := \{F_1^v, F_2^v, \ldots, F_J^v\}$
  - each component is an $M \times N$ matrix of weather force

Error Grids $E := \{E_1, E_2, \ldots, E_J\}$: $M \times N$ errors of each force

- $E_i \in E$ is the error range of $F_i^u$ and $F_i^v$.

$x_{\text{goal}} := (row \in M, col \in N)$: coordinates of goal in $\mathcal{R}$.

## Problem Formulation: Players & State Transition

Players $P := \{P_1 := \text{robot} \; ; \; P_2 := \text{environment}\}$

Stages $\mathbf{K} := \{1...K\}$

Action space for $P_1$: $U := \{U_1^1 \dots U_K^B\}$

- $B$: number of actions - discrete heading angles, and *halt*
- Control action for $P_1$ is $u_k^b \in U$.

Action space for $P_2$: $\Theta := \{\Theta_1^1 \dots \Theta_K^A\}$

- $A$: number of actions - a tuple of selected latitudinal and longitudinal components
- Control action for $P_2$ is $\theta_k^\alpha$

State space $X$

- At each stage $k$, the game state is $x_k$
- Each state has an associated robot location $x_{\text{loc}} := (row \in M, col \in N)$

State transition function $f_k : x_{k+1} = f_k(x_k, u_k, \theta_k^\alpha)$

# Problem Formulation: 2-Player, Zero-Sum Games

Game $G := B \times A$

- Rows $\rightarrow P_1$ action choices
- Columns $\rightarrow P_2$ action choices

$$g_{b,a} := \begin{cases} 0 & \textbf{if } x_{\text{loc}} = x_{\text{goal}} \\ t_{b,a} & \textbf{if } R(x_{\text{loc}}) = 0 \\ D_{\max} & \textbf{if } R(x_{\text{loc}}) = 1 \end{cases}$$

Cost of joint strategy $t_{b,a} := work_{b,a} + cost2go(f_k(x_k, u_k, \theta_k^{\alpha}))$

- $work_{b,a}$: Applied work done by $P_1$
- $cost2go(x_k)$: cost to go to goal after reaching state $x_k$.

$actgrid(x_k)$: $P_1$ action after reaching state $x_k$

$\uparrow$ *the motion plan itself.*

# Algorithms, Quick Look

**Algorithm 1: DynamicPlanner**

- Initialize *cost2go* values to maximum, *actgrid* values to NULL
- For each dynamic programming iteration $I_{DP}$:
  - *cost2go*, *actgrid* ← **DynamicPlannerIter(***cost2go***,** *actgrid***)**
- return *cost2go*, *actgrid*

**Algorithm 2: DynamicPlannerIter**

- $Q \leftarrow$ empty FIFO queue
- Assign $cost2go_{goal} := 0$, $actgrid_{goal} :=$ "halt"
- Add neighbors of $x_{goal}$ to $Q$
- While $Q$ is not empty:
  - $c \leftarrow$ deque $Q$
  - Enqueue neighbors of $c$ that have never been enqueued
  - $cost2go_c$, $actgrid_c \leftarrow$ **NashSolver**$(c)$
- return *cost2go*, *actgrid*

# Algorithms, Quick Look

**Algorithm 3: NashSolver**

- Init $P_1$ mixed policy as $B$-length 0-vector, $P_2$ as $A$-length 0-vector
- $P_1$'s selected action (row) $\leftarrow$ 0, $P_2$'s selected action (column) $\leftarrow$ 0
- For each iteration $I_{nash}$:
  $P_1$ selects counter row to minimize game value on $P_2$'s column
  $P_2$ selects counter column to minimize game value on $P_1$'s row
  $P_1$ increments mixed policy at index specified by row
  $P_2$ increments mixed policy at index specified by column
- $P_1$ action $\leftarrow$ most frequently chosen row
- $P_2$ action $\leftarrow$ most frequently chosen column
- Cost $\leftarrow$ game value at $P_1$ action and $P_2$ action
- return cost (*cost2go*), $P_1$ action (*actgrid*)

Based on 2-player, 0-sum game solver code by Raymond Hettinger:
`code.activestate.com/recipes/496825-game-theory-payoff-matrix-solver`

# Algorithm 1: DynamicPlanner

**Algorithm 1: DynamicPlanner**: Motion planner

**Input:** $x_{\mathrm{goal}}$, $\mathcal{R}$, $D_{\max}$, $I_{DP}$
**Output:** *cost2go*, *actgrid*

Set M $\leftarrow$ number of rows in $R$;
Set N $\leftarrow$ number of cols in $R$;
Initialize grid $cost2go \leftarrow M \times N$, all cells having value $D_{\max}$;
Initialize grid $actgrid \leftarrow M \times N$ **NULL** matrix;
`/* Iteratively update` *cost2go*, *actgrid* `*/`
**for** $i$ *in range* $0 \ldots I_{DP}$ **do**
    `/* Call Algorithm 2 */`
    $cost2go, actgrid \leftarrow$ **DynamicPlannerIter**($x_{\mathrm{goal}}$, $cost2go, actgrid$, $D_{\max}$);
**return** *cost2go, actgrid*

# Algorithm 2: DynamicPlannerIter

**Algorithm 2: DynamicPlannerIter**:
Assigns costs, actions to to each cell in occupancy grid $\mathcal{R}$

**Input:** $x_{\text{goal}}$, *cost2go*, *actgrid*, $D_{\text{max}}$
**Output:** *cost2go*, *actgrid*

```
/* Q: cells that need assignment */
Initialize FIFO queue Q ← ∅;
/* V: remembered all added cells */
Initialize set V ← ∅;
Start at x_goal;
Set cost2go_goal = 0, actgrid_goal = "halt";
Add neighborhood grid cells to Q and to V;
while Q ≠ ∅ do
    Cell c ← Dequeue Q;
    Add neighborhood cells to Q if c not in V;
    Add neighborhood cells to V if c not in V;
    /* Call Algorithm 3 */
    g_{b,a}, P_1^policy, P_2^policy ← NashSolver(c);
    Set cost2go(c) = g_{b,a};
    Set actgrid(c) = P_1^policy;
return cost2go, actgrid
```

# Algorithm 3: NashSolver

**Algorithm 3: NashSolver**:
Approximate, iterative 0-sum 2-player game solver

**Input:** $G$, $I_{\text{nash}}$
**Output:** cost $g_{b,a}$, policies $P_1^{\text{policy}} \in U$ and $P_2^{\text{policy}} \in \Theta$

$B \leftarrow$ number of rows in $G$;
$A \leftarrow$ number of columns in $G$;
`/* Record whenever action selected */`
Init $P_1^{\text{policy}} \leftarrow B$-length 0-vector;
Init $P_2^{\text{policy}} \leftarrow A$-length 0-vector;
`/* Init actions as first row, col */`
Set $P_1^a$, $P_2^a$ action $\leftarrow 0$;
**for** *i in range* $0 \dots I_{nash}$ **do**
    $P_1^a \leftarrow {}_b(G(b, P_2^a)), b \in B$ ;
    $P_2^a \leftarrow {}_a(G(P_1^a, a)), a \in A,$ ;
    `/* Increment action count */`
    $P_1^{\text{policy}}[P_1^a]1;$
    $P_2^{\text{policy}}[P_2^a]1;$
$P_1^{\text{policy}} \leftarrow P_1^{\text{policy}} / I_{\text{nash}};$
$P_2^{\text{policy}} \leftarrow P_2^{\text{policy}} / I_{\text{nash}};$
**return** $P_1^{policy}$, $P_2^{policy}$

# Theorems

**Theorem 1.- Global Optimum Convergence**

After a finite number of dynamic programming iterations, the motion plan converges to a global optimum. The maximum number of executions is proportional to the region dimensions.

Large regions will be a large planning search space

**Theorem 2.- Single-Iteration Feasibility**

A single iteration of dynamic programming creates an unobstructed plan to the goal from any reachable cell. Each terminates in a deterministic, finite number of iterations.

Each iteration a feasible motion plan: intermediate solution

# Computational Complexity: Big-O Analysis

$$\mathcal{O}^{NASH} = \mathcal{O}((B + A) \times I_{nash})$$
$$\mathcal{O}^{DPI} = \mathcal{O}(M \times N \times \mathcal{O}^{NASH})$$

Substituting $\mathcal{O}^{NASH}$

$$\mathcal{O}^{DPI} = \mathcal{O}(M \times N \times ((B + A) \times I_{nash}))$$

Also

$$\mathcal{O}^{DP} = \mathcal{O}(I_{DP} \times \mathcal{O}^{DPI})$$

Substituting $\mathcal{O}^{DPI}$

$$\mathcal{O}^{DP} = \mathcal{O}(I_{DP} \times (M \times N \times ((B + A) \times I_{nash})))$$

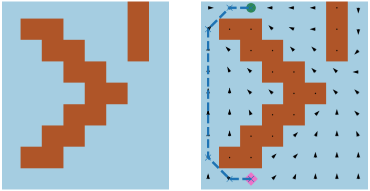
**Very high complexity!**

- Can intermediate solutions give usable motion plans?

# Experimental Results

# Simple Region Results – Synthetic Data

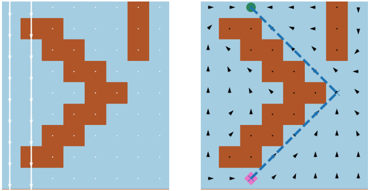
No water currents



**Goal:** green circle
**Start:** pink diamond

- No currents → shortest path

Certain currents that oppose boat

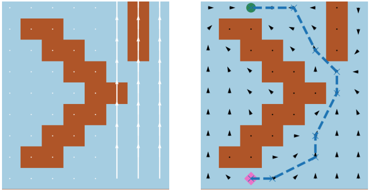


Current direction is directly against the boat

- Boat chooses a longer path to save energy

# Simple Region Results – Synthetic Data

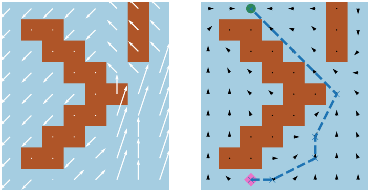
Certain currents that help boat



Currents directed towards goal

- Boat chooses a longer path to ride the currents

Uncertain, antagonistic currents

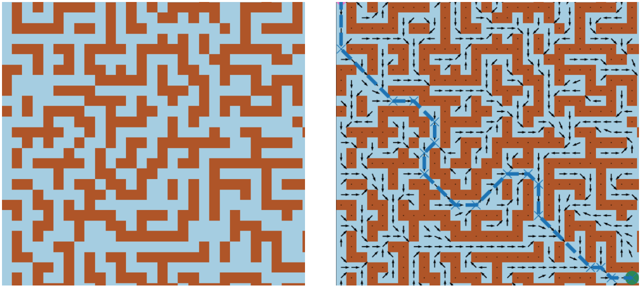


Antagonistic weather modifies currents to oppose boat

- Error range: bounds weather
- Ride currents when it can, shortest-path when it cannot

# Maze Region Results – Synthetic Data

Maze with no water currents

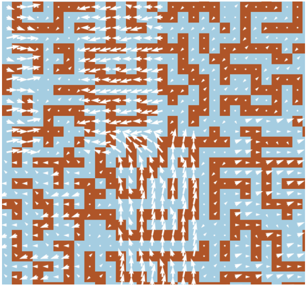


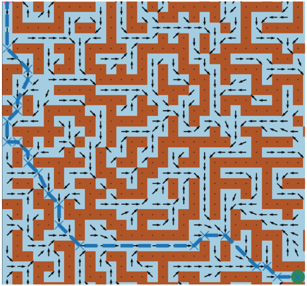
Region Motion Plan

Dynamic programming gives optimal maze solution

# Maze Region Results – Synthetic Data

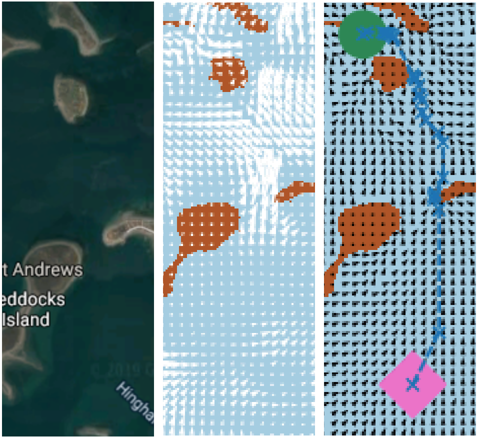
Maze with antagonistic water currents





Currents can have a dramatic impact on best solution

# Marine Region – Real Data Results



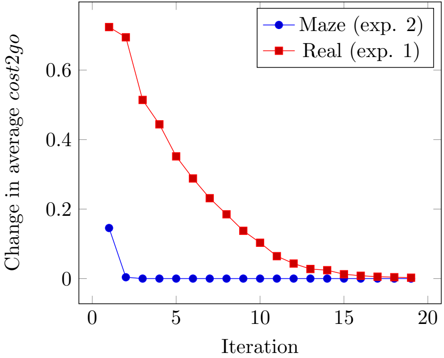

Massachusetts Bay region

- Northeast Coastal Ocean Forecast System (NECOFS)
- Antagonistic currents

Plan Shown: 20 iterations

- Strategic planning evident

# Dynamic Programming Convergence



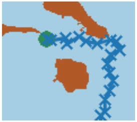

Real marine region

- Convergence shows very little improvement in average cost after only 20 iterations
- Feasibility of onboard planning despite high complexity

# Comparison with Particle Swarm Optimization

- PSO: 500 iterations (no uncertainty)
- DP: dynamic programming, 20 iterations (no uncertainty)
- GTDP: game theory dynamic programming, 20 iterations

PSO



DP

GTDP

Each path applied to both the certain and worst-case scenarios

| Solution | Work, static forces | Work, antagonistic forces |
|---|---|---|
| *PSO* | 294349 | 320234 |
| *DP* | 345085 | 368574 |
| *GTDP* | 297142 | 319969 |

Blue cells indicate the scenario used for generating that solution

# Conclusions

# Conclusions & Future Work

## Conclusions

- Game theory allows robot to handle worst-case weather
- Real data suggests boat can benefit from strategic planning
- High complexity offset by ability to use early iterations
- Online forecasts such as NECOFs enable better autonomous navigation

## Future Work

- Incorporate dynamic water currents
- Incorporate dynamic vehicle model
- Dynamically consider currents too strong for boat
- Extend to 3D (underwater and aerial applications)

# Conclusions & Future Work

## Towards a Real Robotic Boat

- Building airboat for shallow-water applications
- Modified Zelos ProBoat and EMILY ERS

End of Presentation

**Game Theoretic Potential Field for Autonomous Water Surface Vehicle Navigation Using Weather Forecasts**

Questions?