# Lecture 1: Introduction to Networked Control Systems

**Richard M. Murray**
Caltech Control and Dynamical Systems
16 March 2009

**Goals:**

- Describe current and emerging applications of networked control systems
- Summarize key features and recent advances in NCS
- Provide an overview of the course contents

**Reading:**

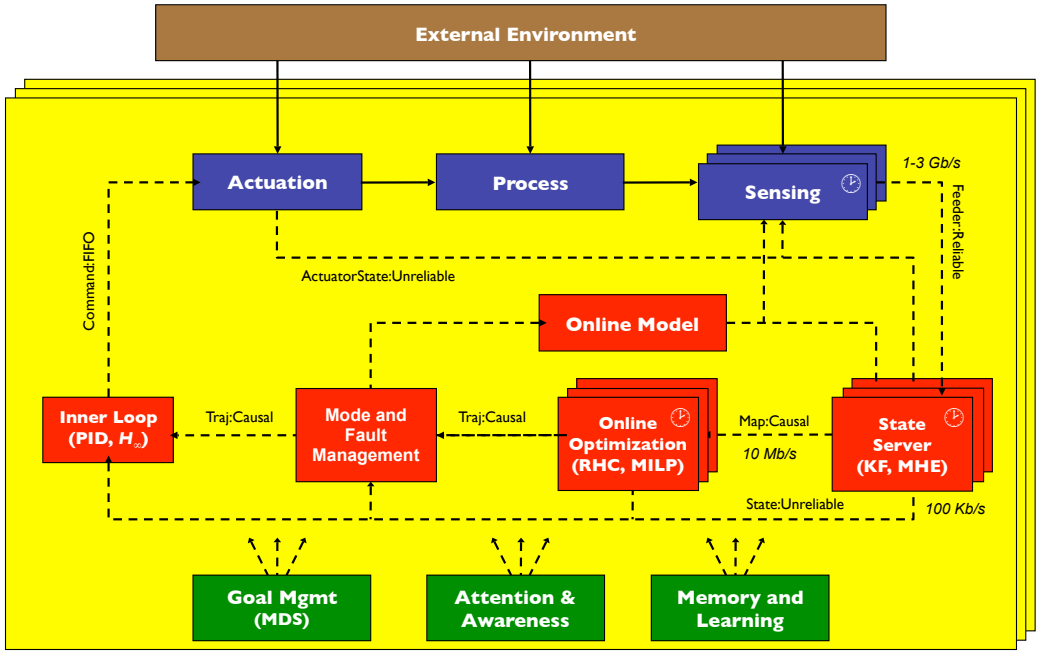
- Networked Control Systems, 2008 (preprint). Chapter 1
- Control in an Information Rich World, Sections 1, 3.2 and 3.3

Available on course wiki page

http://www.cds.caltech.edu/~murray/wiki/ncs-sp09

## Networked Control Systems

(following P. R. Kumar)



# Applications



# Team Caltech: Alice

**Team Caltech**

- Started in 2003, for DGC04
- 2004-05: 50 Caltech undergraduates, 1 MS student, 3 TAs, 2 faculty

**Alice**

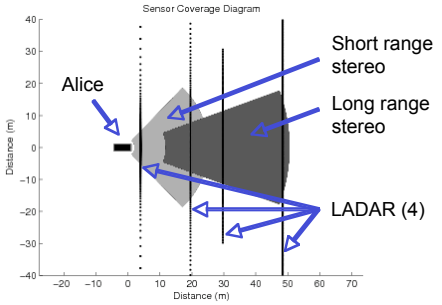
- 2005 Ford E-350 Van
- 5 cameras: 2 stereo pairs, roadfinding
- 5 LADARs: long, med*2, short, bumper
- 2 GPS units + 1 IMU (LN 200)

**Computing (2005)**

- 6 Dell PowerEdge Servers (P4, 3GHz)
- 1 IBM Quad Core AMD64 (fast!)
- 1 Gb/s switched ethernet

**Software**

- 15 programs with ~100 exec threads
- 100,000+ lines of executable code

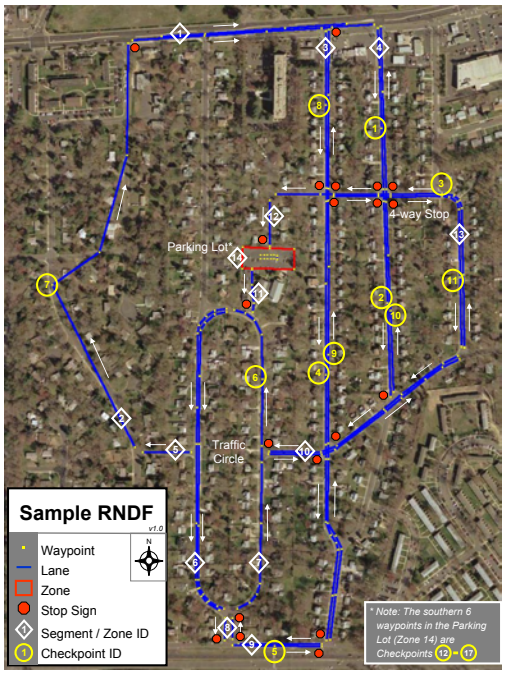


# 2007 DARPA Grand Challenge (Urban Challenge)

**Autonomous Urban Driving**

- 60 mile course, less than 6 hours
- City streets, obeying traffic rules
- Follow cars, maintain safe distance
- Pull around stopped, moving vehicles
- Stop and go through intersections
- Navigate in parking lots (w/ other cars)
- U turns, traffic merges, replanning
- Prizes: $2M, $1M, $500K



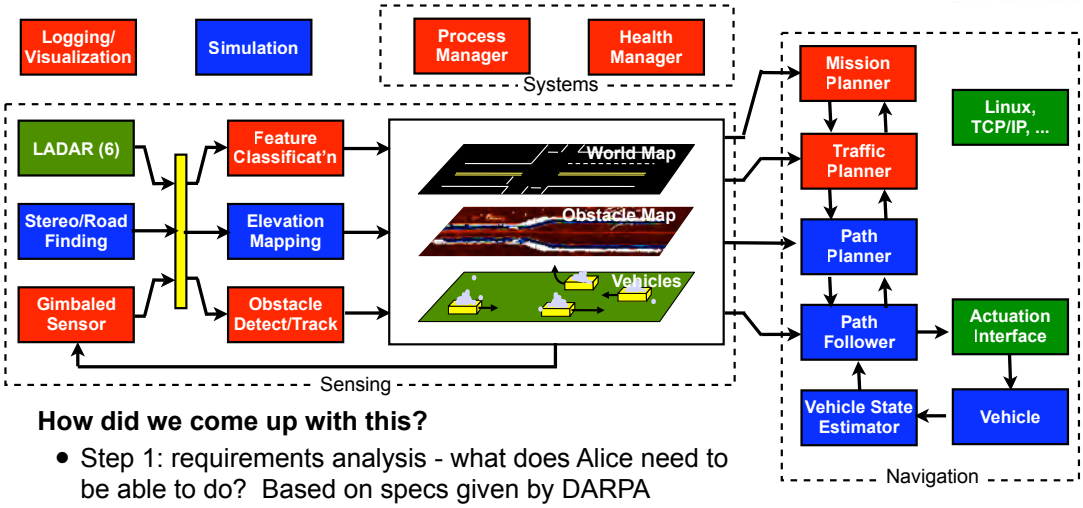






# Urban Driving




# DGC07 System Architecture




**How did we come up with this?**

- Step 1: requirements analysis - what does Alice need to be able to do? Based on specs given by DARPA
- Step 2: functional decomposition - what are the basic elements required to function? Designer choice
- Step 3: scenario generation and iteration - can it do what we want? Some simulation; mainly paper-based
- Step 4: interface specs (50% inherited ⇒ software reuse)

**Properties**

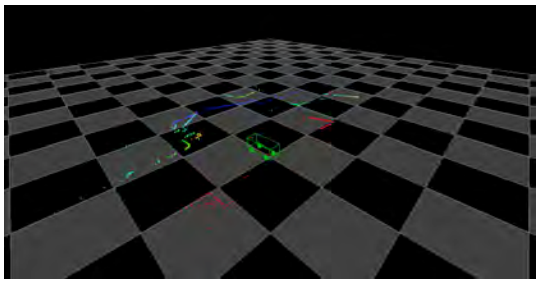
- Highly modular
- Rapidly adaptable
- Constantly viable
- Robust ???

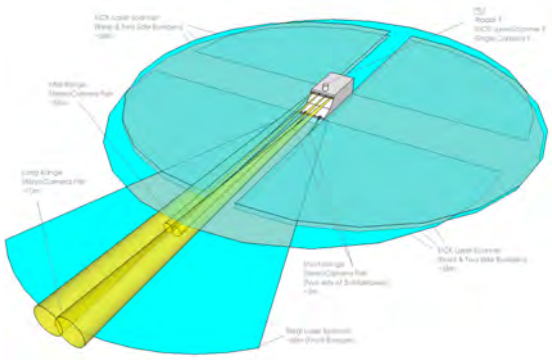


# Sensing System

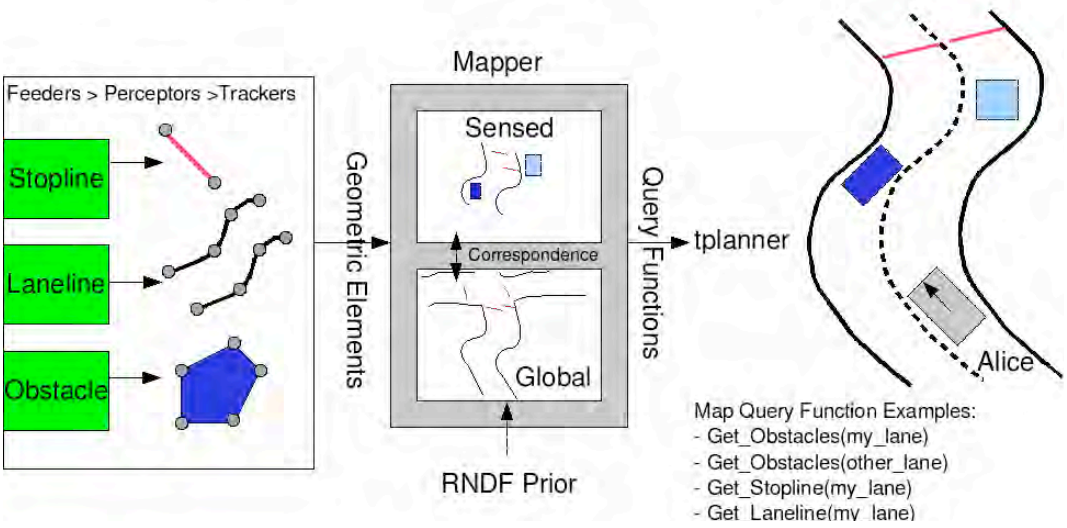
**Sensing hardware**

- 6 horizontal LADAR (overlapping)
- 1 pushbroom LADAR; 1 sweeping (PTU)
- 3 stereo pairs (color; 640x480 @ ~10 Hz)
- 2 road finding cameras (B&W)
- 2 RADAR units (PTU mounted)
- 10 blade cPCI high speed computing





# Sensor Processing




**Feeders**

- Interface to raw sensor data

**Perceptors**

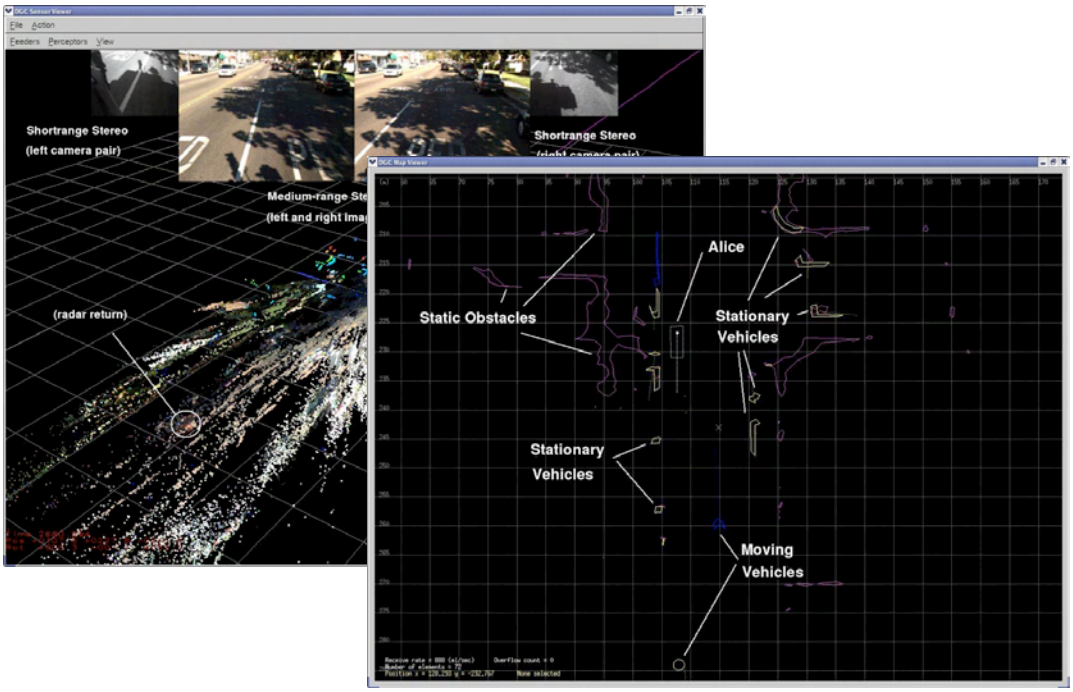
- Detect and track features needed for driving

**Mapper**

- Maintain locations of current objectives



# Feeder → Perceptors → Mapper



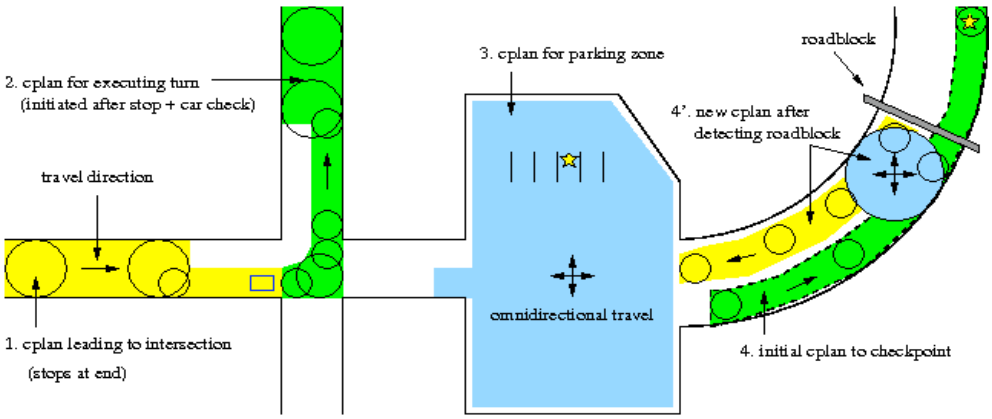
# Mission and Traffic Planners

**Mission Planner performs high level decision-making**

- Graph search for best routes; replan if routes are blocked

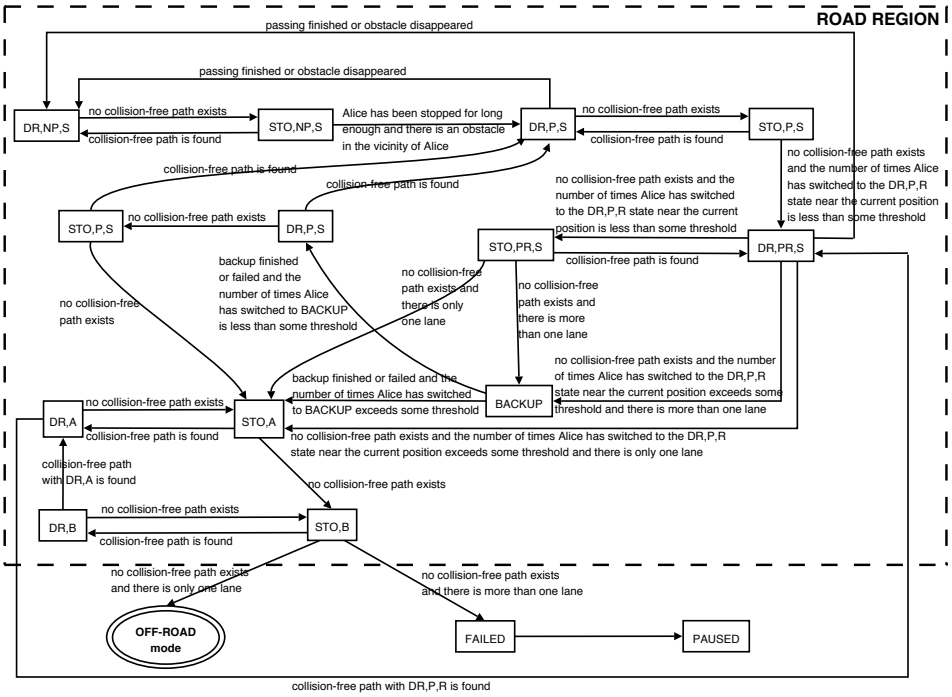
**Traffic Planner handles rules of the road**

- Control execution of path following & planning (multi-point turns)
- Encode traffic rules - when can we change lanes, proceed thru intersection, etc
- Coordinate vehicle avoidance strategies (passive, active avoidance)






# Logic Planner



# Trajectory Generation in Real-Time

**Key obstacle: constraints**

- Optimization routines slow down significantly when constraints are added

$$u_{[t,t+\Delta T]} = \arg\min \int_t^{t+T} L(x(\tau),u(\tau))\,d\tau + V(x(t+T))$$

$$x_0 = x(t) \quad x_f = x_d(t+T)$$

$$\dot{x} = f(x,u) \qquad g(x,u) \le 0$$

**Technique #1: flatness (Fliess et al)**

- Exploit structure of the dynamics to replace differential constraints

$$z = h(x, \dot{x}, \ldots, q^{(p)})$$
$$x = x(z, \dot{z}, \ldots, z^{(q)})$$
$$u = x(z, \dot{z}, \ldots, z^{(q)})$$

- Trajectory generation → linear eqs

$$x(t_0) \to (z(0), \dot{z}(0), \ldots, z^{(p)}(0))$$
$$x(t_0) \to (z(0), \dot{z}(0), \ldots, z^{(p)}(0))$$

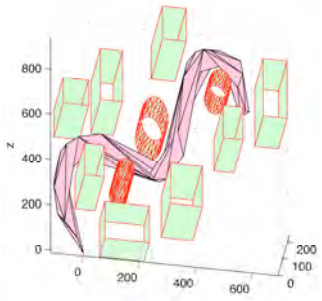
$$z = \sum \alpha_i \phi^i(t) \qquad M\alpha = \begin{bmatrix} \bar{z}(t_0) \\ \bar{z}(t_f) \end{bmatrix}$$

**Technique #2: NURBs (Flores/Milam)**

- Idea: choose basis functions so that constraints (on flat vars) are *automatically* satisfied

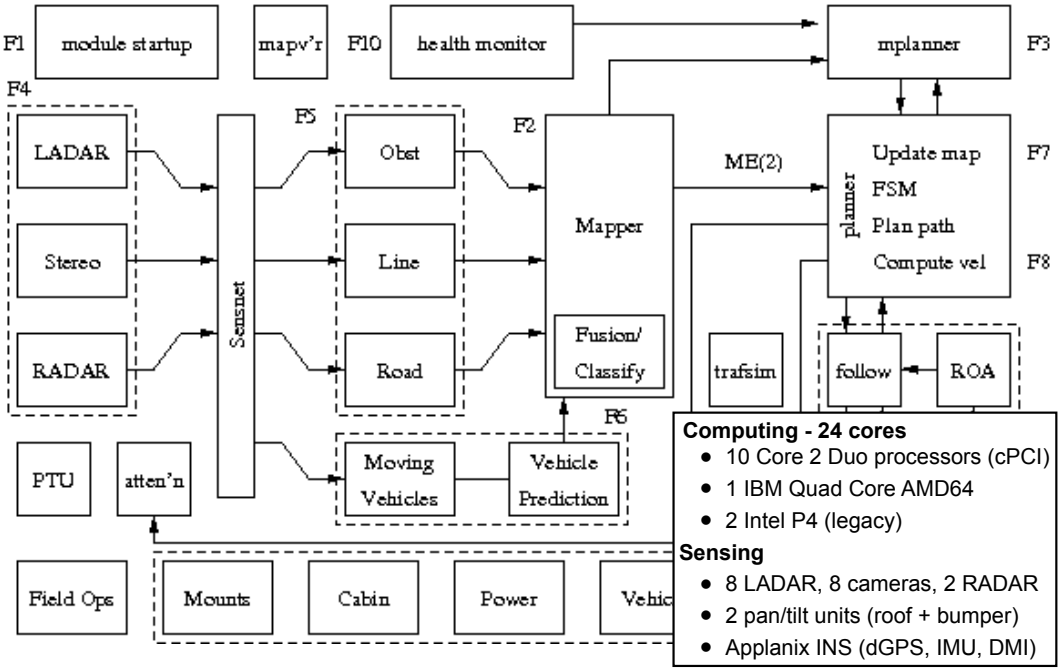
$$z(t) = \sum R^i(t) P_i \quad R^j(t) = \frac{B_j(t) w^j}{\sum_k p B_k(t) w^k}$$

- $z(t) \in$ convex hull of control points $P_i$





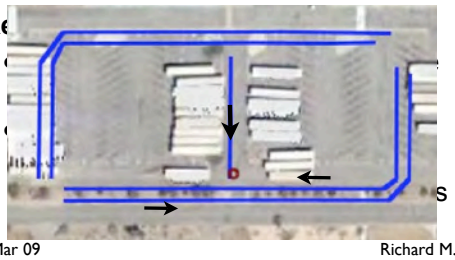

# Architecture, July 2007



# 2007 National Qualifying Event

**Merging test**

- 10-12 cars circling past inters'n
- Count "perfect runs" in 30 min








# 2007 National Qualifying Event

**Driving test**

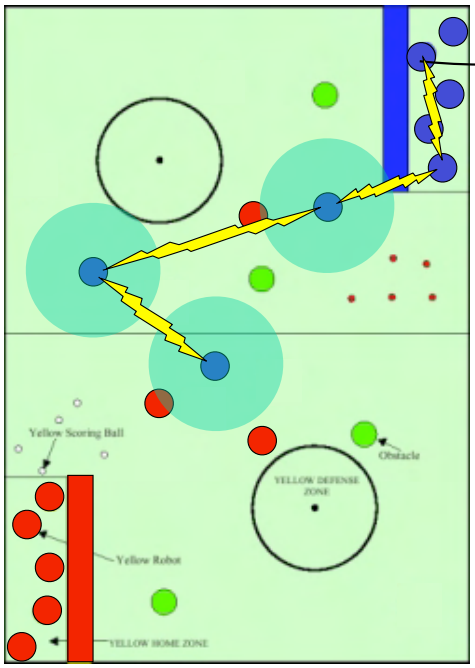
- 2 mile run - roads, parking lots, obstacles on road

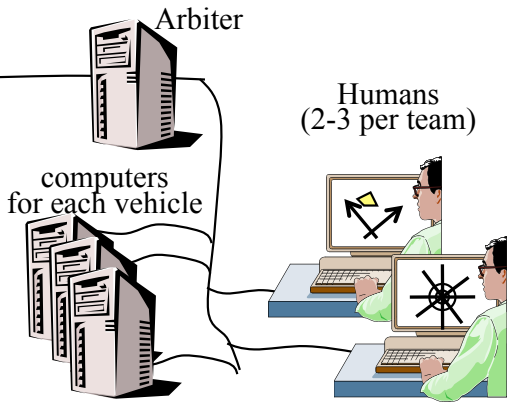
**Results**

- First run - safety buffers too large => slow progress
- Second run - completed course in 22 minutes; minor errors
- 1 of ~8 vehicles completed

D'Andrea & M
ACC 2003

# Example: RoboFlag (D'Andrea, Cornell)



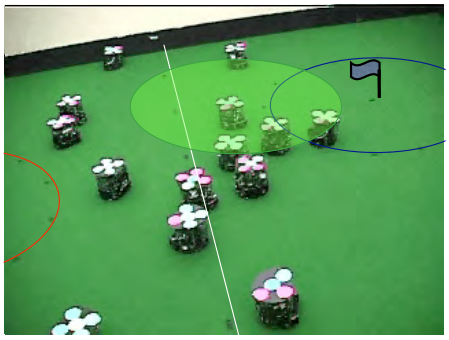

**Robot version of "Capture the Flag"**

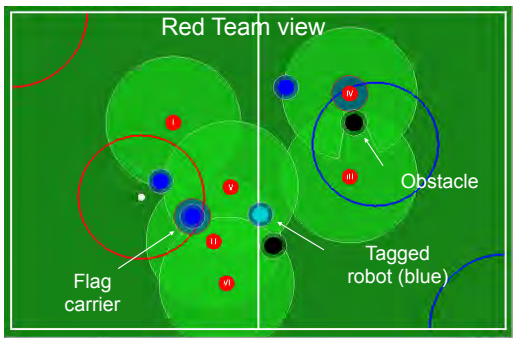
- Teams try to capture flag of opposing team without getting tagged
- Mixed initiative system: two humans controlling up to 6-10 robots
- Limited BW comms + limited sensing



Hayes et al
ACC 2003

# RoboFlag Demonstration





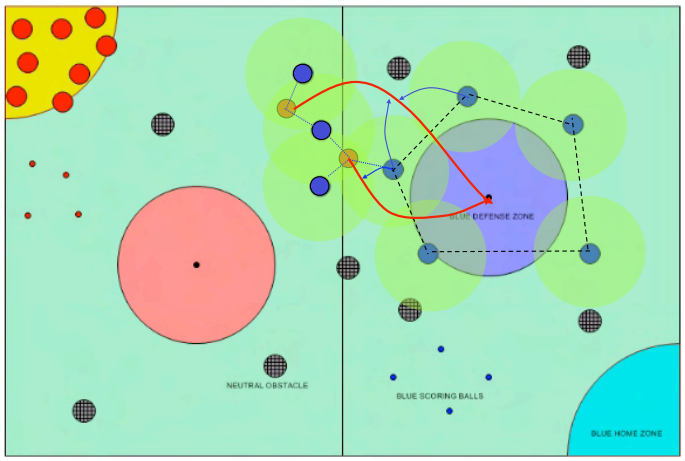

**Integration of computer science, communications, and control**

- Time scales don't allow standard abstractions to isolate disciplines
- Example: how do we maintain a consistent, shared view of the field?

**Higher levels of decision making and mixed initiative systems**

- Where do we put the humans in the loop? what do we present to them?
- Example: predict "plays" by the other team, predict next step, and react

# RoboFlag Subproblems




**1.Formation control**
- Maintain positions to guard defense zone

**2.Distributed estimation**
- Fuse sensor data to determine opponent location

**3.Distributed consensus**
- Assign individuals to tag incoming vehicles

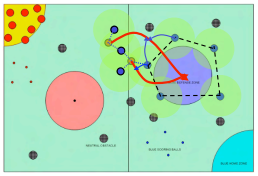
**Goal: develop systematic techniques for solving subproblems**

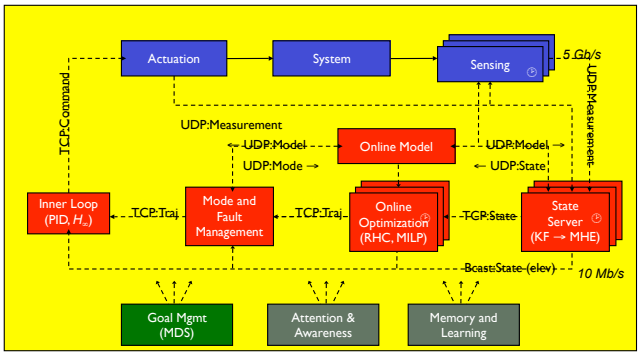
- Cooperative control and graph Laplacians
- Distributed estimation and sensor fusion
- Distributed receding horizon control
- Packet-based estimation and control
- Verifiable protocols for consensus and control

→ Implement and test as part of annual RoboFlag competition



# Control Problems and Design Patterns






**Control Challenges**
- How should we distribute computing load burden between computers?
- How should we handle communication limits and dropped packets?
- How do multiple computers cooperate in a shared task (with common view)?
- What types of protocols should we use for transmitting data between nodes?

**Design Patterns**
- Local temporal autonomy - allow modules to operate with data losses
- State estimation - estimate future states if current data are not available
- Control buffers - buffer commands to tolerate latency and lost data
- Time servers - time stamp data and track clock skew