CDS 110a: Course Project Lecture #2
Vehicle Control for Alice

Richard M. Murray

5 April 2006

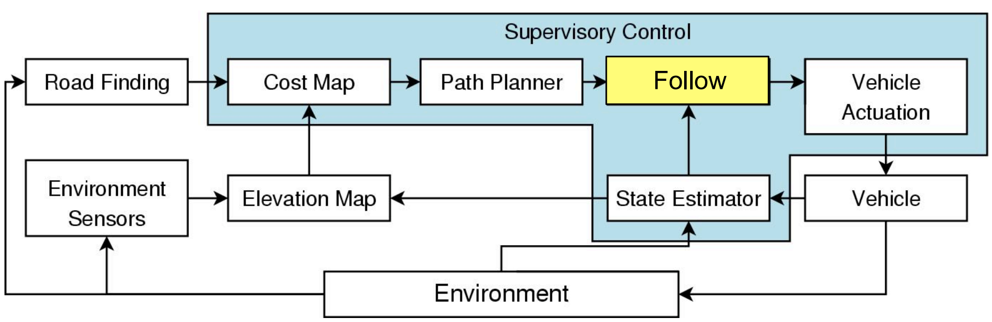
Goals:

- Describe how the trajectory tracking controller for Alice works
- Highlight open issues and possible course projects

Reading:

- "Asynchronous Network Control of Multiple Low Bandwidth Devices using Linux", T. Foote. E11 paper, 2006

Control System Specification



Controller Specification

- 50 cm transient error (overshoot)
- 20 cm steady state error (noise)

Method: discrete time, state space ctrl

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k + Du_k$$

- Set $u = (x_d, \hat{x})$; use B to subtract

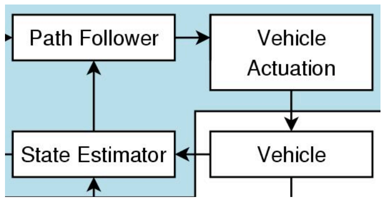
Inputs

- Reference trajectory from path planner
- Current state estimate (pos, vel, acc) from state estimator
- Disturbances from environment (unmeasured)

Outputs

- Normalized steering, throttle and brake commands (velocity and accel/decel)

Vehicle Actuation



Adrive

- HW: steering, throttle, brake, ignition, transmission, engine diagnostics - serial port interfaces
- In: normalized actuation commands, engine diagnostics (OBD II)
- Out: actuator values and engine state
- Independent threads for each actuator
- “Interlock” logic to ensure safety

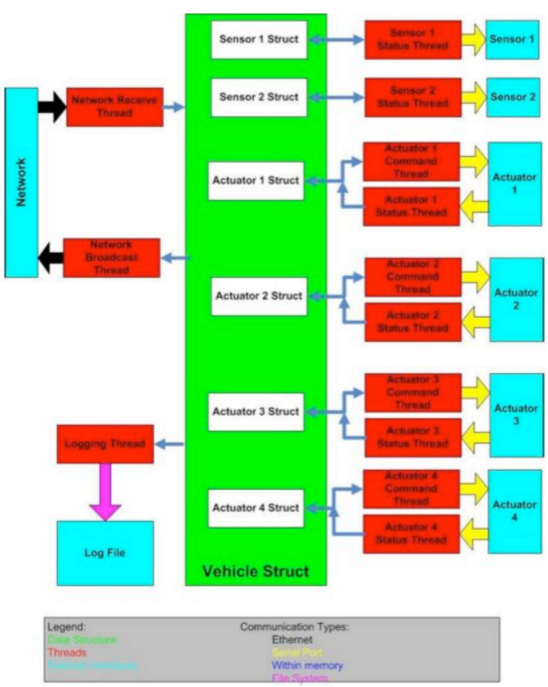
Actuator command

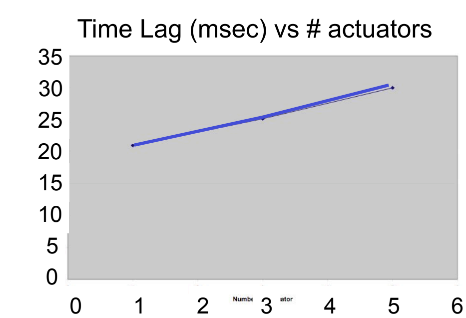
- *Commands sent individual to actuators*
- Actuator: steer, accel, gas, brake, estop, trans
- Command: set position, vel, acc
- Argument: double or string

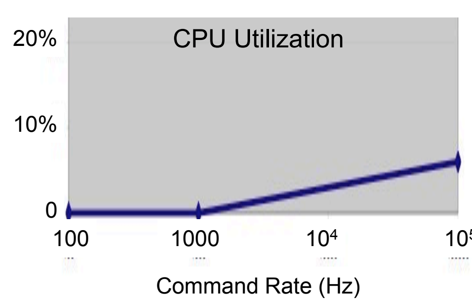
Actuator state (30 Hz)

- Steering: status, pos, cmd, update time
- Gas: status, pos, cmd, update time
- Brake: status, pos, cmd, pressure, update
- Estop: status, darpa, adrive, software, update time, "about to pause"
- Trans: status, cmd, pos, update_time
- OBD II: status, engine RPM, time since start, wheel speed, coolant temp, wheel force, glow plug lamp time, throttle position, gear ratio, update time

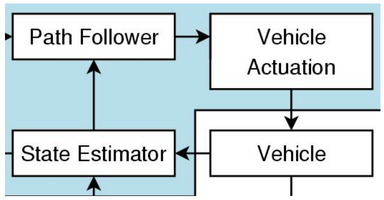
Adrive Performance







Vehicle State

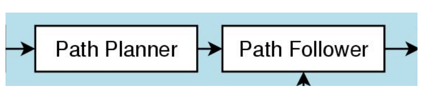


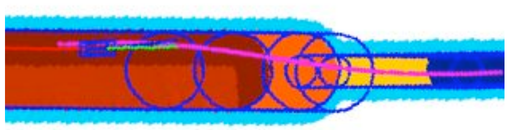
Astate
- HW: 2 GPS units (2-10 Hz update), 1 inertial measurement unit (gryo, accel @ 400 Hz)
- In: actuator commands, actuator values, engine state
- Out: time-tagged position, orientation, velocities, accelerations
- Use vehicle wheel speed + brake command/position to check if at rest

State message (40 Hz)
- Timestamp (microseconds)
- Northing, easting, altitude (meters)
- Roll, pitch, yaw (radians)
- Velocity and acceleration for above
- Confidence levels for position and orientation (variance)

StateClient
- Provides class that automatically grabs state info via spread
- Allows interpolation of state estimates for finer resolution

Path Planner



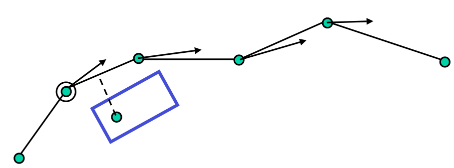


PlannerModule
- HW: none
- In: speed maps, vehicle state
- Out: desired trajectory
- Algorithm runs on quadcore AMD64 at approx. 5 Hz

RDDFpathgen
- Generates paths based on route definition (RDDF)
- Straight line interpolation between waypoints

Trajectory (~5 Hz, async)
- numPoints - number of points in traj
- order - number of derivatives
- N[numPoints*order], E[…] - traj points and derivatives
- minSpeed - slowest speed along traj

Accessor functions
- getClosestPoint - get index of nearest point on on trajectory
- interpolate - point, vel, acc of nearest point on trajectory



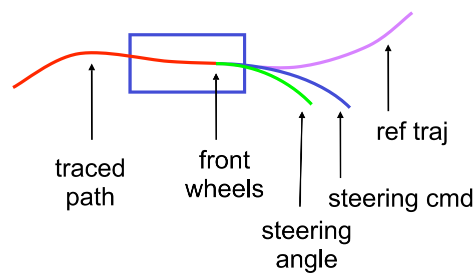
Alice Infrastructure

Skynet
- Wrapper for spread; provides standard functions
- Each process is a skynet "module"; modules define spread groups
- Uses FIFO message type (FIFO by sender, reliable)
- Logging and playback capability

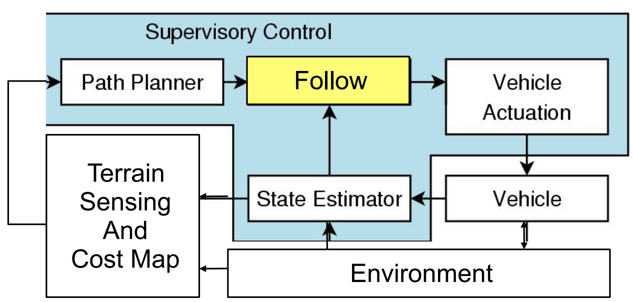
Sparrow
- Real-time user interface library
- Allows display of internal program variables in real-time
- Allows users to set variable values, execute actions that control operation
- Works across simple terminal interface

DGCutils
- Get current time (microseconds)
- Mutex and condition interfaces
- Thread safe sleep

GUI
- Listens to all spread messages
- Provides display of elevation maps, cost maps, planned trajectories, etc

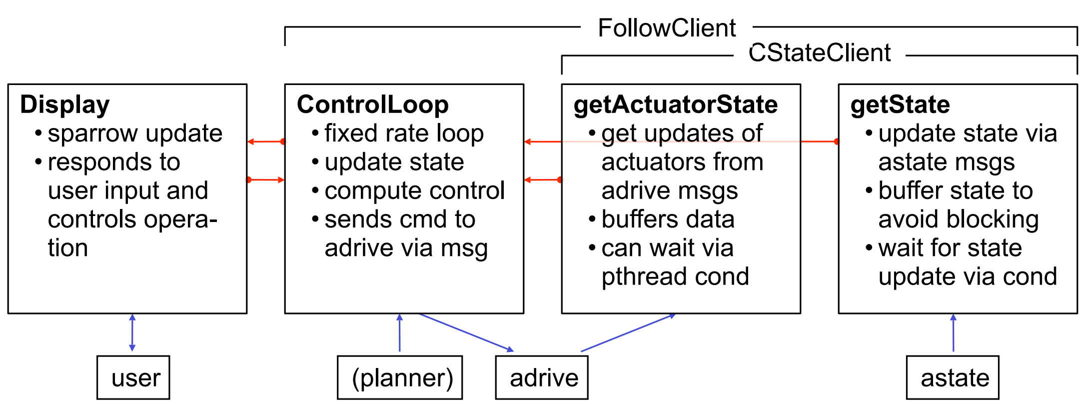


Follow



Vehicle Dynamics

$$
\begin{aligned}
\dot{N} &= v \cos\theta \\
\dot{E} &= v \sin\theta \\
\dot{\theta} &= \frac{v}{L} \tan\phi \\
\dot{\phi} &= \omega = u_1 \\
\dot{v} &= a = u_2
\end{aligned}
$$



Main Program

```
main() {
  // Process command line arguments

  // Initalize skynet module
  pSkynetkey = getenv("SKYNET_KEY");
  sn_key = atoi(pSkynetkey);
  client = new FollowClient(sn_key);

  // NB: starts StateClient threads

  // Start member threads
  DGCstartMemberFunctionThread(
    client,
    &FollowClient::ControlLoop);

  // Start display
  dd_open();
  dd_usetbl(maindisp);
  dd_bindkey(…);  // key bindings
  dd_loop();      // user interface
  dd_close();

  return 0;
```

Comments

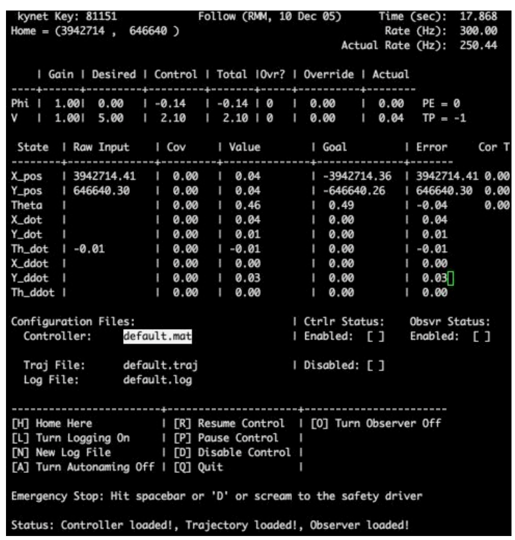
- FollowClient is a derived class of StateClient
  - StateClient automatically creates threads for reading from astate and adrive on creation
- StateCliient is a derived class of CSkynetContainer
  - CSkynetContainer contacts the spread server and starts a “heartbeat” thread to send messages to “SNmodlist group”
- Additional FollowClient initialization loads controller from files, initializes parameters, etc
- ControlLoop thread is standard pthread
- Remaining thread handles sparrow display

Thread 1: Real-Time Display (Sparrow)

maindisp.dd

- Defines the main sparrow display

```
%%
Skynet Key: %key              Follow (RMM, 10 Dec…
Home = (%xorigin, %yorigin)   Rate (Hz):  %rate
                         Actual Rate (Hz):  %arate

    | Gain | Desired | Control | Total |Ovr?
  ----+------------+-----------------
    +-----------------+-------------+----------
Phi | %pGn | %pFF    | %pCntrl | %pCmd | %pOv|
V   | %vGn | %vFF    | %vCntrl | %vCmd | %vOv|

%%

short:  %key sn_key "%3d" -ro;
double: %rate controlRate "%5.2f";
double: %arate actualRate "%5.2f" -ro;
double: %time currentTime "%7.3f" -ro;

double: %xorigin xorigin "%7.0f";
double: %yorigin yorigin "%7.0f";
…
button: %PAUSE "Pause" pauseControl;
…
tblname: maindisp;
```



- ‘cdd’ compiler turns .dd file into .h file that defines ‘maindisp’ table
- dd_loop() updates screen and accepts user input

Thread 2: Control Computation

```
while (1) {
  DGCgettime(usecStart);

  UpdateState();
  UpdateActuatorState();

  DGCgettime(timeNow);
  currentTime = DGCtimetosec(timeNow-timeStart);
  traj_read(m_traj_falcon, trajVector, currentTime);

  outCtrl = ss_compute(m_lateralController, inp);
  outCmd[PHI] = outGain[PHI]*(outCtrl[PHI] + outFF[PHI]);

  steer_Norm = outCmd[PHI]/VEHICLE_MAX_AVG_STEER;
  steer_Norm = fmax(fmin(steer_Norm, 1.0), -1.0);

  my_command.my_actuator = steer;
  my_command.number_arg = steer_Norm;
  m_skynet.send_msg(m_adriveMsgSocket, &my_command, ...);

  DGCgettime(usecStop);
  if(numMicroSecTotal > (usecStop - usecStart))
```

- Infinite loop
- Get start time
- Update state
- Determine reference value
- Compute control using Falcon
- Normalize cmd
- Send command to adrive
- Sleep to end of cycle

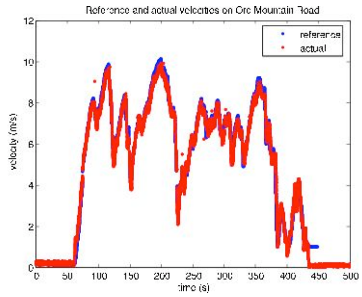
Thread 3, 4: State Client

```
void CStateClient::getActuatorStateThread() {
  int actuatorstatesocket =
    m_skynet.listen(SNactuatorstate, ALLMODULES);
  while(m_bRunThreads) {
    if(m_skynet.get_msg(actuatorstatesocket,
      &m_rcvdActuatorstate, sizeof(m_rcvdActuatorstate), 0,
      &pActuatorstateMutex) != sizeof(m_rcvdActuatorstate))
        skynet_error();
    DGCSetConditionTrue(condNewActuatorState);
  }
}

void CStateClient::UpdateActuatorState()
{
  DGClockMutex(&m_actuatorstateMutex);
  memcpy(&m_actuatorState, &m_rcvdActuatorstate, sizeof(…));
  DGCunlockMutex(&m_actuatorstateMutex);
}

void CStateClient::WaitForNewActuatorState() {
  DGCWaitForConditionTrue(condNewActuatorState);
  UpdateActuatorState();
  condNewActuatorState.bCond = false;
}
```

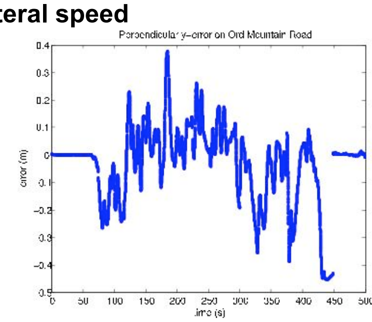
- Thread to read msgs
- Infinite loop
- Read msg (blocks until available)
- Unblock any-one waiting
- Copy state into buffer
- Use mutex to insure completeness
- Block until new state msg arrives

TrajFollower Performance

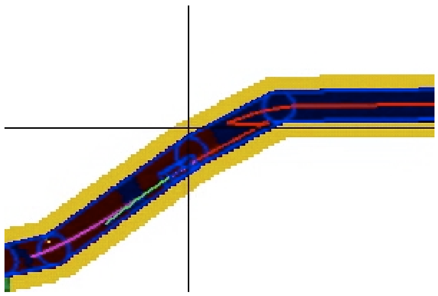
Longitudinal speed



Lateral speed



GPS jumps



Kalman Filter