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

Reading:
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

**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**
- 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
Adrive Performance

Command Rate (Hz)

100
1000
10^4
10^5

CPU Utilization

0%
10%
20%

Time Lag (msec) vs # actuators

0
5
10
15
20
25
30
35

0  1  2  3  4  5  6

Legend:

Communication Types: Ethernet

Within memory

File System

Sensor 1 Struct
Sensor 2 Struct
Actuator 1 Struct
Actuator 2 Struct
Actuator 3 Struct
Actuator 4 Struct
Log File

Network Receive Thread
Network Broadcast Thread
Logging Thread

Sensor 1 Status Thread
Sensor 2 Status Thread
Actuator 1 Command Thread
Actuator 2 Command Thread
Actuator 3 Command Thread
Actuator 4 Command Thread
Actuator 1 Status Thread
Actuator 2 Status Thread
Actuator 3 Status Thread
Actuator 4 Status Thread
Vehicle State

**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

**Astate**
- HW: 2 GPS units (2-10 Hz update), 1 inertial measurement unit (gyro, 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
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 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

DGutils
- 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

Supervisory Control

Path Planner

Follow

Vehicle Actuation

State Estimator

Vehicle

Terrain Sensing And Cost Map

Environment

Vehicle Dynamics

\[
\begin{align*}
\dot{N} &= v \cos \theta \\
\dot{E} &= v \sin \theta \\
\dot{\theta} &= \frac{v}{L} \tan \phi \\
\phi &= \omega = u_1 \\
v &= a = u_2
\end{align*}
\]

Display

- sparrow update
- responds to user input and controls operation

ControlLoop

- fixed rate loop
- update state
- compute control
- sends cmd to adrive via msg

getActuatorState

- get updates of actuators from adrive msgs
- buffers data
- can wait via pthread cond

getState

- update state via astate msgs
- buffer state to avoid blocking
- wait for state update via cond

FollowClient

CStateClient
Main Program

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

    // Initialize 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 a state and a drive on creation

• StateClient 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
<table>
<thead>
<tr>
<th>Gain</th>
<th>Desired</th>
<th>Control</th>
<th>Total</th>
<th>Ovr?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phi</td>
<td>%pGn</td>
<td>%pFF</td>
<td>%pCntrl</td>
<td>%pCmd</td>
</tr>
<tr>
<td>V</td>
<td>%vGn</td>
<td>%vFF</td>
<td>%vCntrl</td>
<td>%vCmd</td>
</tr>
</tbody>
</table>
```

- `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) {
    DGC_gettime(usecStart);
    UpdateState();
    UpdateActuatorState();
    DGC_gettime(timeNow);
    currentTime = DGC_gettime_usec(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, ...);
    DGC_gettime(usecStop);
    if(numMicroSecTotal > (usecStop - usecStart))
        usleep(numMicroSecTotal - (usecStop - usecStart));
}
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();
            DGCSSetConditionTrue(condNewActuatorState);
    }
}

void CStateClient::UpdateActuatorState() {
    DGCMlockMutex(&m_actuatorstateMutex);
    memcpy(&m_actuatorState, &m_rcvdActuatorstate, sizeof(...));
    DGCUunlockMutex(&m_actuatorstateMutex);
}

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

- Thread to read msgs
- Infinite loop
- Read msg (blocks until available)
- Unblock anyone 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
Integrated System
# Course Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Topic</th>
<th>Lecturers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction and motivating examples</td>
<td>Murray</td>
</tr>
<tr>
<td>2</td>
<td>Embedded systems programming</td>
<td>Murray</td>
</tr>
<tr>
<td>3</td>
<td>Trajectory generation, RHC</td>
<td>Murray, Keviczky</td>
</tr>
<tr>
<td>4</td>
<td>State estimation (KF, MHE)</td>
<td>Sandberg, Cremean</td>
</tr>
<tr>
<td>5</td>
<td>Packet-based estimation and control</td>
<td>Sinopoli (Stanford)</td>
</tr>
<tr>
<td>6</td>
<td>Packet-based estimation and control</td>
<td>Mostofi, Shi</td>
</tr>
<tr>
<td>7</td>
<td>Distributed estimation and control</td>
<td>Gupta</td>
</tr>
<tr>
<td>8</td>
<td>Cooperative Control</td>
<td>Jin, Keviczky</td>
</tr>
<tr>
<td>9</td>
<td>Project presentations</td>
<td>You!</td>
</tr>
</tbody>
</table>

- Project proposals due 2 May (~end of week 5; sooner is OK!)

**No class on Friday**