CS 378 - Autonomous Vehicles in Traffic I

Week 5a - Software Architecture
Today

- We'll revisit the software sections of the paper in Reading Assignment 1.
- We'll talk about Road Network Description Files (RNDF) and Mission Description Files (MDF)
- We'll discuss the major components of our software architecture and how they are interconnected. These include commander, navigator, pilot, maplanes and observers.
- We'll also take a look at the simulator for the car.
Mobile robot navigation

For any kind of autonomous navigation, you need to know a map of the world. Such a map can be provided in a number of formats.

- For robot soccer, our map is hard-coded because of the structured nature and limited size of the environment.
- For indoor navigation, the map is generally unstructured and typically contains information on what locations contain obstacles.
- For road navigation, the map is highly structured. You need information about roads, intersections, number of lanes, lane widths etc.
Map of soccer field in Robotics Lab
Map of PRC

What is the best way to represent this information?
Route Network Definition File (RNDF)

- RNDFs were used by DARPA to provide information about the road network for the Urban challenge.
- We still use RNDFs for navigation. Using the data provided by the Applanix POS-LV, we can generate our own RNDF.
- RNDFs provide information about the following: way-points, lanes, stop-signs, zones and parking spots.
- RNDFs serve as a rich source of information, which is not available for commercial roads as of yet.
- Let us take a look at specific portions of a sample RNDF.
  - Sample RNDFs can be found in art_map/rndf
  - The next few slides use prc_large.rndf
RNDF

RNDF_name       prc_large.rndf
num_segments    6
num_zones       1
format_version  1.0
creation_date   23-Sep-07

/* Road D */
segment 1
num_lanes       2
lane    1.1
num_waypoints   15
checkpoint      1.1.5   2
checkpoint      1.1.10  15
stop    1.1.6
exit    1.1.6   5.1.1
...  1.1.1   30.385707       -97.731700
...  1.1.6   30.384024       -97.728322
1.1.7   30.383928       -97.728134
1.1.8   30.383668       -97.727607
...  1.1.8   30.383668       -97.727607
... end_lane
... /

/* Harry Ransom Trail */
segment 5
num_lanes       2
lane    5.1
num_waypoints   8
...  5.1.1   30.384066       -97.728147
5.1.2   30.384315       -97.728040
...  5.1.2   30.384315       -97.728040
... end_lane
RNDF - *lanes, waypoints, checkpoints*

```
RNDF_name       prc_large.rndf
num_segments    6
num_zones       1
format_version  1.0
creation_date   23-Sep-07

/* Road D */
segment 1
num_lanes       2
lane 1.1
num_waypoints   15
checkpoint      1.1.5   2
checkpoint      1.1.10  15
stop    1.1.6
exit    1.1.6   5.1.1
... 

1.1.1   30.385707       -97.731700
... 
1.1.6   30.384024       -97.728322
1.1.7   30.383928       -97.727607
1.1.8   30.383668       -97.726607
... 
end_lane
... 

/* Harry Ransom Trail */
segment 5
num_lanes       2
lane 5.1
num_waypoints   8
... 

5.1.1   30.384066       -97.728147
5.1.2   30.384315       -97.728040
... 
end_lane
```
RNDF - exits

RNDF_name prc_large.rndf
num_segments 6
num_zones 1
format_version 1.0
creation_date 23-Sep-07

/* Road D */
segment 1
num_lanes 2
lane 1.1
num_waypoints 15
checkpoint 1.1.5 2
checkpoint 1.1.10 15
stop 1.1.6
exit 1.1.6 5.1.1
...
1.1.1 30.385707 -97.731700
...
1.1.6 30.384024 -97.728322
1.1.7 30.383928 -97.728134
1.1.8 30.383668 -97.727607
...
end_lane
...
/* Harry Ransom Trail */
segment 5
num_lanes 2
lane 5.1
num_waypoints 8
...
5.1.1 30.384066 -97.728147
5.1.2 30.384315 -97.728040
...
end_lane

Harry Ransom Trail
Road D
RNDF - zones

RNDF_name       prc_large.rndf
num_segments    6
num_zones       1
format_version  1.0
creation_date   23-Sep-07

... zone 7
num_spots       2
perimeter       7.0
exit 7.0.2 1.2.4
exit 7.0.2 1.1.13
exit 7.0.7 1.2.2
num_perimeterpoints 12
7.0.1 30.382885 -97.725831
7.0.2 30.382855 -97.725735
7.0.3 30.382837 -97.725675
7.0.4 30.382766 -97.725721
7.0.5 30.382603 -97.725292
7.0.6 30.382710 -97.725211
7.0.7 30.382692 -97.725211
7.0.8 30.382674 -97.725168
7.0.9 30.382515 -97.725279
7.0.10 30.382540 -97.725422
7.0.11 30.382616 -97.725685
7.0.12 30.382730 -97.725843
end_perimeter
spot 7.1
checkpoint 7.1.2 17
7.1.1 30.382627 -97.725437
7.1.2 30.382648 -97.725430
end_spot
spot 7.2
checkpoint 7.2.2 18
7.2.1 30.382688 -97.725594
7.2.2 30.382707 -97.725591
end_spot
end_zone

This visualization does not show the entire zone, but just the connections to the spots.
### RNDF - zones

<table>
<thead>
<tr>
<th>RNDF_name</th>
<th>prc_large.rndf</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_segments</td>
<td>6</td>
</tr>
<tr>
<td>num_zones</td>
<td>1</td>
</tr>
<tr>
<td>format_version</td>
<td>1.0</td>
</tr>
<tr>
<td>creation_date</td>
<td>23-Sep-07</td>
</tr>
</tbody>
</table>

... zone 7

| num_spots       | 2             |
| perimeter       | 7.0           |
| exit            | 7.0.2 1.2.4   |
| exit            | 7.0.2 1.1.13  |
| exit            | 7.0.7 1.2.2   |

<table>
<thead>
<tr>
<th>num_perimeterpoints</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0.1 30.382885 -97.725831</td>
<td></td>
</tr>
<tr>
<td><strong>7.0.2 30.382855</strong> -97.725735</td>
<td></td>
</tr>
<tr>
<td>7.0.3 30.382837 -97.725675</td>
<td></td>
</tr>
<tr>
<td>7.0.4 30.382603 -97.725292</td>
<td></td>
</tr>
<tr>
<td>7.0.5 30.382700 -97.725581</td>
<td></td>
</tr>
<tr>
<td><strong>7.0.6 30.382692</strong> -97.725211</td>
<td></td>
</tr>
<tr>
<td>7.0.7 30.382674 -97.725168</td>
<td></td>
</tr>
<tr>
<td>7.0.8 30.382515 -97.725279</td>
<td></td>
</tr>
<tr>
<td>7.0.9 30.382540 -97.725422</td>
<td></td>
</tr>
<tr>
<td>7.0.10 30.382616 -97.725685</td>
<td></td>
</tr>
<tr>
<td>7.0.11 30.382730 -97.725843</td>
<td></td>
</tr>
<tr>
<td>end_perimeter</td>
<td></td>
</tr>
</tbody>
</table>

| spot 7.1          |
| checkpoint 7.1.2 17 |
| 7.1.1 30.382627 -97.725437 |
| **7.1.2 30.382648** -97.725430 |
| end_spot          |

| spot 7.2          |
| checkpoint 7.2.2 18 |
| 7.2.1 30.382688 -97.725594 |
| **7.2.2 30.382707** -97.725591 |
| end_spot          |

end_zone
Mission Description File (MDF)

- Specifies which checkpoints you have to reach in this mission
- This MDF describes a small loop in PRC. It is the loop we run by default when we test code at the car.
Maplanes

• Maplanes parses RNDFs to generate a drivable map. This is done in a number of steps:
  ◦ It first parses the RNDF into a graph structure
  ◦ The graph is used to generate smooth curves (C1 Hermite curves)
  ◦ Generates polygons from these smooth curves
• The polygons comprise drivable surface that Navigator tries to drive over.
• Maplanes was initially constructed to be a temporary substitute to real-time road sensing but was found to be fairly effective.
  ◦ Real-time road sensing is a very difficult problem. Few research labs use this as a primary means of navigation.
Maplanes - How it works
Maplanes - Accuracy
Maplanes - polygons

Road D

Transition Polygons

Harry Ransom Trail
Maplanes - The art_map package

The code for Maplanes and RNDF data structure lies in the art_map package

• If you are interested, you can start tracing the code in src/nodes/mapl.cc.
• This package also includes the code for parsing RNDFs, creating graphs, smooth curves and polygons.
• This package also contains a library to work with polygons called PolyOps.
• The package launches a single ROS node called maplanes during autonomous runs.
• More information about the package is available at:
  ◦ http://www.ros.org/wiki/art_map
Maplanes - The *maplanes* node

- Publishes the following messages:
  - Polygons:
    - /roadmap_global [art_msgs/ArtLanes] - all polygons are published when the node is initialized
    - /roadmap_local [art_msgs/ArtLanes] - all polygons within a distance of 80 meters from the car
  - Visualization Messages:
    - /visualization_marker [visualization_msgs/Marker]
    - /roadmap_cloud [sensor_msgs/PointCloud]
    - /visualization_marker_array [visualization_msgs/MarkerArray]
- Subscribes to odometry:
  - /odom [nav_msgs/Odometry] - the current position and velocity of the car published by the applanix
- You can view this information for any node by using the command `rosnode info <node-name>` (for instance `rosnode info /maplanes`)
Applanix

• The Applanix driver is in the *applanix* package in the *art_vehicle* stack.
• Obtains the GPS position and IMU values from the Applanix POS-LV
• Publishes the position and velocity of the car on the following topic:
  - `/odom` [nav_msgs/Odometry]
Navigation - Commander and Navigator

- The main navigation code is split up into 2 functional units:
  - **Commander**
    - Performs route planning.
    - Reads RNDF and MDF, performs the A* algorithm and works out the way-points that need to be visited to go through all the checkpoints
  - **Navigator**
    - Performs the task of reaching the next waypoint determined by commander.
    - Decides the actions that need to be executed by *pilot*
    - Has a ton of code!
- This abstraction is similar to that of *global* and *local* planning in navigation.
Navigator

- At the top level, Navigator is a hierarchical state machine
- At the highest level it has 4 states:
  - Run
  - Pause
  - Disable
  - Suspend
- Computes the best action (steering, velocity) at each time step.
- This action is then translated into velocity and steering.
- Responsible for dealing with readings from observers

State machine for the RUN state
The *art_nav* package

- All the code for commander and navigator is contained inside the *art_nav* package.
- This package contains the "intelligence" behind our vehicle.
- The 2 main nodes are *commander* and *navigator*.
- The code is in the *include* and *src* directories.

- The commander subscribes to the navigator state:
  - `/navigator/state` [*art_msgs/NavigatorState*]
- It publishes out commands to the navigator:
  - `/navigator/cmd` [*art_msgs/NavigatorCommand*]
The *navigator* node

- Publishes the following messages:
  - /pilot/cmd [art_msgs/CarDriveStamped]
  - /navigator/state [art_msgs/NavigatorState]
  - /ioadr/cmd [art_msgs/IOadrCommand]

- Subscribes to the following messages:
  - /navigator/cmd [art_msgs/NavigatorCommand]
  - /odom [nav_msgs/Odometry]
  - /roadmap_local [art_msgs/ArtLanes]
  - /ioadr/state [art_msgs/IOadrState]
  - /observations [art_msgs/ObservationArray]
Pilot

- Implements the low-level control for the car
- Converts requested velocity and steering into actuator commands
- Uses a number of PID controllers for this
  - PID controllers can help you choose how much throttle to give when you want to move from a speed of say $2 \, \text{m/s}$ to $5\, \text{m/s}$
- Controls the throttle, brake, steering and gear-shifter
- The pilot node can be found in the `art_pilot` package
- The `throttle`, `ioadr`, `brake` and `steering` nodes can be found in the `art_servo` package. The shifter is controlled through the IOADR board. Look in this package to see how we interface with the actuators on the car.
Pilot - Messages

- pilot -> throttle: /throttle/cmd [art_msgs/ThrottleCommand]
- throttle -> pilot: /throttle/state [art_msgs/ThrottleState]

- pilot -> brake: /brake/cmd [art_msgs/BrakeCommand]
- brake -> pilot: /brake/state [art_msgs/BrakeState]

- pilot -> steering: /steering/cmd [art_msgs/SteeringCommand]
- steering -> pilot: /steering/state[art_msgs/SteeringState]

- pilot -> shifter: /shifter/cmd [art_msgs/Shifter]
- shifter -> pilot: /shifter/state [art_msgs/Shifter]
Observers

• Re-implemented in ROS last semester as a student project.
• We have *forward*, *backward* and *adjacent* lane observers working.
• Observers work by calculating which polygons in their area of interest are occupied using data from the Velodyne.
• Each observer figures out its area of interesting by selecting the corresponding polygons from */roadmap_local*
• Observers estimate both *position* and *velocity* of obstacles and forward this information to the Navigator
• Observers also estimate whether it is safe to move ahead given our current velocity
nearest_forward at work
The *observers_node* ROS node

• A single node that contains all the observers

• Publishes observations at:
  ○ /observations [art_msgs/ObservationArray]

• Subscribes to the following messages:
  ○ /roadmap_local [art_msgs/ArtLanes]
  ○ /odom [nav_msgs/Odometry]
  ○ /velodyne_obstacles [sensor_msgs/PointCloud]

• We will talk about finding obstacle positions through the Velodyne next week
Putting it all together - *graph in simulation!*

- If you have the code running, you can generate this graph using the command `rxgraph`
- In simulation, *artstage* is subscribed to the actuator states to execute it in the simulator. Additionally it provides "fake" sensor information by publishing `/velodyne_obstacles` (Velodyne) and `/odom`, `/imu` (Applanix)
Using my MS Paint skills - *graph at the car!*

- I was planning on grabbing this image from *rxgraph* at the car, but then we switched lectures.
- The main difference is that the Applanix and Velodyne driver now publish the sensor information.
- An important point to note is that *most* other nodes are *exactly* the same. Apart from the *art_servo* ones, they don't know when we are running a simulation. This aids us when we test out our code.
Running the simulator

• Use the tutorials on the wiki to obtain our code-base and setup the ROS environment on a machine.
  ◦ You will need to do this for Programming Assignment 2

• Once you have our code-base, build it using the `rosmake` command and then launch the simulator
  ◦ `rosmake art_run`
  ◦ `roslaunch art_run auto_stage.launch`

• The `art_run` package contains scripts to run all the required nodes as once. We'll come back to it when we see `roslaunch`. 
Review

• Today we went through the software of our car at a high level.
• You should refer to these slides when you are doing your projects and want to figure out where the other pieces of our code-base are.

• This lecture concludes the introductory part of the course.
• We'll now move on to sensor processing. Next week (Wednesday), I hope to show you some fun applications of point cloud processing.