MISSION PLANNER AND CONTINGENCY FRAMEWORK FOR RANGER

Heather Pinnix
Nekton Research, LLC

ABSTRACT:

One of the goals of the Ranger MicroUUV program is to develop an inexpensive AUV for multi-vehicle distributed robotics applications. To achieve this, we developed a flexible mission planner and contingency framework to handle unexpected events using a wide range of sensors and mission structures.

Multi-agent missions demand that vehicles make decisions based on multiple sets of data. This observation underlies the design of Ranger. The software framework is built around the idea that Ranger needs to take independent actions based on its own sensor readings and the readings of other vehicles.

Ranger's software is state-based. A state encapsulates all the tasks needed to complete a goal. Goals are decision points for the vehicle. Therefore a state may have multiple goals as long as the goals are not contradictory. This allows both vehicle cooperation and unilateral action.

This paper describes the state structure, mission setup and task integration of Ranger.

Introduction

In 2001 Nekton Research, LLC was awarded a contract from DARPA to design and deploy a lost asset and survivor location system (LSALS) using our Ranger vehicle in partnership with Sandia National Labs.

By nature, autonomous vehicles must respond to continuous flow of information, i.e. they must process sensor data and make decisions based on past information (filters), current information (sensor triggers), and expectations of future information (safety checks). Ranger's intelligence firmware manages these three types of data.

Encapsulation was designed into Ranger from the beginning to separate the need to understand what operations are performed from how they are performed. To achieve this, we designed a four-layer architecture. The layers are drivers, behaviors, states and missions.

Each of the four layers hides the mechanics from the functionality, allowing a programmer to focus on the design if the mission rather than the details of the implementation.

Ranger Mission Structure

Language

The firmware structure is platform-independent. However, the implementation needs to take into account limitations of hardware. The processing power of a DSP is superior to that of PC104's but they do not support a wide variety of languages. We use C++ because it allows us to use Object Orientation, create re-usable modules for our DSP. C++ support is provided by TI's Code Composer Studio.

Driver Layer: Primitives and Sensors

Functions which directly access the hardware are called drivers. Drivers that directly control the physical vehicle are called primitives. Another set of functions, sensor drivers, process and control the sensors. Primitives differ from sensors because they physically alter the vehicle.

Behavior Layer

Ranger is event-driven. Events fall in to three categories: timers, incoming sensor data, and safety triggers. Threads are used to synchronize events to actions. Behaviors hide the mechanics of threads from the user. Generally, but not always, behaviors are registered to a sensor. When a behavior is registered to a sensor, the event of incoming sensor data triggers the thread. The mechanism to retrieve, store, and process the necessary data is defined within each driver.

An example of this is the yaw controller behavior. This behavior receives data at 10Hz from the compass sensor driver. The behavior retrieves the desired heading, processes the data, stores the yaw actuator position, and finally calls the setYawServo primitive function.

State Layer

Depending on Ranger's goals, different behaviors need to run. The state encapsulates a collection of goals pertaining to what Ranger is attempting to accomplish. It is responsible for starting the appropriate behaviors, monitoring them to insure that no errors have occurred, and checking to see if the goals have been met. Lastly the state stops the behaviors when the state expires.

The gotoXYZ state for example uses seven behaviors: Navigation Filter, Guidance Controller, Yaw Controller, Depth Controller, Pitch Controller, Timer, and Porpoising. The behaviors are initialized at state startup. When the state exits either because of an error, success, or a signal from the mission layer, the state stops the all seven of its behaviors, and exits.

Mission Layer

The mission layer is the highest level of control of Ranger's firmware. By selecting the active state the mission layer controls the desired goals. The mission layer also starts global behaviors, i.e. fail-safe checks and other behaviors, unrelated to any state.
As an example, the LSALS mission uses three states, Wait State, LSALS Algorithm State, and an Exit State. The mission runs the wait state first. When the wait state exits, the mission starts the LSALS State. After the LSALS State exits, the LSALS mission starts the Exit State. The Exit State is terminal; the only way to exit this state is by a user terminating the LSALS mission.

Communications

In this architecture, communication is implemented as a behavior. To handle interaction between the user and the Ranger fleet, we developed a Fleet Control Software package (FCS). Data is transferred from the FCS in the following way (Figure 1): Ranger surfaces, receives a valid GPS fix, and sends a data packet to FCS. Ranger then sends a ready packet. If a command for that vehicle is pending, FCS retrieves and transmits it. If data packets have been received for this vehicle since the last time it surfaced, FCS forwards them. When all data has been received by the vehicle, FCS sends a resume command and Ranger dives to the desired depth.

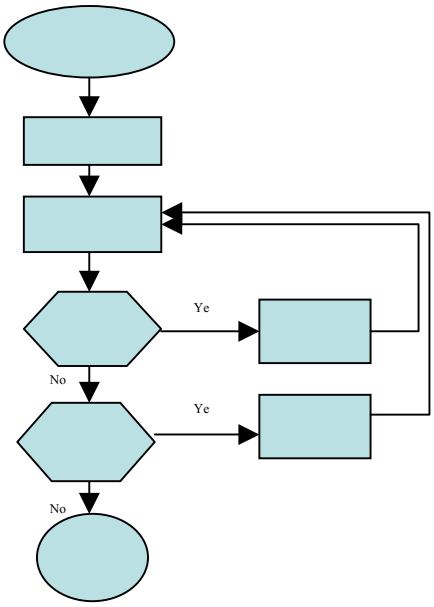

Figure 1: FCS protocol

LSALS Mission

As mentioned earlier, the mission that test the LSALS algorithm has three states which we now describe (Figure 2).

Figure 2: LSALS mission.

Wait State:

The wait state's objective is to hold the vehicle in limbo until all vehicles are deployed. During the wait state, Ranger polls the FCS server once a second for any commands. The wait state exits when Ranger receives a deploy command.

LSALS State:

The objective of the LSALS state is to find the source of a pseudo-plume using an algorithim that was developed by Sandia Labs. This

algorithm takes sensor data from all vehicles and from it computes waypoints for Ranger. The algorithm produces convergence for all vehicles to the center of the plume. To meet the state's objectives, three LSALS-specific behaviors are started: LSALS data, LSALS calculation, and RF transfer.

LSALS Data Behavior

LSALS data behavior creates pseudo-sensor data at a frequency of 10Hz. The sensor value is calculated using the quadratic equation:

$$\frac{1275}{R^2 + 5}$$

where R is the distance from the theoretical plume source.

LSALS Calculation Behavior

The best and worst samples received from the LSALS data behavior are added once per second to an ordered list ranked by sensor value (Figure 3). The current position and pseudo-sensor values are placed first in a 20 sample-long array. To complete the array, the ordered list is manipulated to acquire 19 other unique samples.

The input to the LSALS algorithm is the array of 20 samples. The output of the algorithm is NED coordinates where the algorithm calculates the plume origin and a quality factor.

The three dimensional version of the algorithm is a called first. If the quality factor is less than a certain threshold, the output coordinate is given to the guidance and depth controller, otherwise the two dimensional version of the algorithm is used. If the 2-D version's quality factor is less than a certain threshold the output coordinate is given to the guidance controller, otherwise no new coordinate is set.

When four consecutive values of the vehicle's position are within 0.5m of each other, it is assumed the algorithm has found the plume source and the LSALS state is signaled to exit.

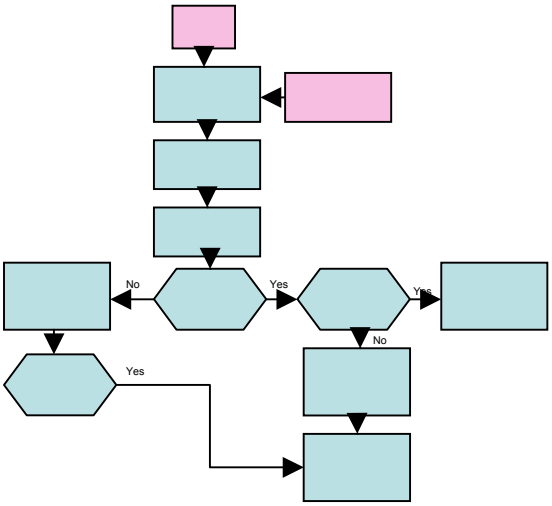

Figure 3: LSALS calculation behavior

RF Transfers Behavior

The RF transfers behavior handles RF communications. A timer is set for 30 seconds. When the timer expires, Ranger's desired depth is set to zero. A valid GPS fix tells the behavior that the antenna is out of the water. The RF communication transfer then proceeds. The data packet contains five samples sent to the FCS. Sample one is the current location and sensor value. The other four samples are evenly spaced between the current sample and the last update. Data packets received from other vehicles go directly into the ordered list used by the LSALS calculation behavior.

Exit State

The objective of the exit state is to send data to the other vehicles and to the FCS after the algorithm has been completed. The assumption is that Ranger has found the plume or the LSALS State has timed out. In either case the vehicle has its motor turned off, and because is it slightly positively buoyant will eventual float to the surface. To accomplish its goals, the exit state uses the RF transfer behavior and LSALS data behavior.

Exit state is a terminal state, and is active until the mission is stopped.

Discussion

We track the resources consumed by each behavior when a state is created. This allows the system to keep a real time model. The state layer gives us the ability view the multi-threading events that drive the mission path in linear manner. The modularity and encapsulation of this architecture simplifies the setup and completion of complex missions. Detailed discussion of the LSALS mission is described in a companion paper [3].

The architecture has successfully been applied to two other missions. During a joint demonstration with NOAA, four Rangers mapped a salinity front at the inlet of the New Bern River. The results of the NOAA tests are reported in [3]. It is also being used in an ongoing demonstration of a multi-agent mine reacquisition algorithm. Because the architecture is modular, we have been able to implement these new missions with minimal modifications to the code, i.e. only mission-specific behaviors and states..

The next round of improvements will focus on giving the mission layer more control over the states. Currently, the mission layer does very little monitoring of the active state. If a problem occurs within a state, the mission aborts. We propose to achieve this by deactivating the current state when an error condition occurs, and activating a contingency state that solves the condition.

Acknowledgments

This work was funded by DARPA under contract DE-AC04-94AL85000. We thank the Nekton LSALS team: Mathieu Kemp, Ryan Moody, and Bryan Schulz, and John Hurtado, Ray Byrne, and Steven Eskridge from Sandia National Lab..

References:

1. A. J. Healey, D. B. Marco, "Current Developments in Underwater Vehicle Control and Navigation: The NPS ARIES AUV". Proceedings of IEEE Oceans 2000, Providence, RI, September 2000.
2. Roy M. Turner, "Controlling Long-Range, Intelligent Autonomous Underwater Vehicles for Ocean Science Research", AAAI Workshop on AI Technologies for Environmental Applications, 1994.
3. Hobson et al., 2003, Multi-UUV Missions using Ranger MicroUUVs, UUST 2003