INDIVIDUAL LAB REPORT 11

Progress Review - 12

Shubham Garg
TEAM H (PHOENIX)

Teammates:
Akshit, Parv, Steve

November 19, 2019

Individual Contribution:

1. Developing and testing mission scripts for UGV using behaviour tree framework
2. Improved robustness of door detection

Developing and testing mission scripts for UGV using behaviour tree framework:

Behavior trees define how a set of actions and conditions should be used to accomplish a task. The tree is made up of execution nodes, control flow nodes, and decorator nodes. Action nodes and condition nodes are the two types of execution nodes. These nodes are where the state of the system is checked and actions are performed. A condition node returns either SUCCESS or FAILURE to indicate what the state of some part of the system is. Behavior tree for our UGV is shown in Figure 1.
Action nodes are used to make the system perform some action if they are active. Action nodes are shown as square nodes in Figure 1. They can be either active or inactive. If a node is active this means the behavior tree has decided to perform the action. If the node is inactive the behavior tree is not trying to perform the action. For example, if husky needs to drive-off, then the "Driveoff" action should be active, while the "Detectdoor" action should be inactive. An active action node can have a SUCCESS, RUNNING, or FAILURE status. SUCCESS indicates that the action has is done being performed and finished successfully, FAILURE indicates the action is done but has failed, RUNNING indicates that the action is still being performed.
Control flow nodes determine which condition nodes are checked and which action nodes are active or inactive. There are three types of control nodes currently implemented:

- **Fallback Nodes:** This node returns FAILURE if and only if all of its children return FAILURE. If one of its children returns RUNNING or SUCCESS, it returns RUNNING or SUCCESS and no subsequent children's statuses are checked. These are shown with a ? in Figure 1.

- **Sequence Nodes:** This node returns SUCCESS if and only if all of its children return SUCCESS. If one of its children returns RUNNING or FAILURE, it returns RUNNING or FAILURE and no subsequent children's statuses are checked. These are shown with a $\rightarrow$ in the figure above.

- **Parallel Nodes:** This node has N children. It returns SUCCESS if M of these children return SUCCESS, for some M N. It returns FAILURE of N - M + 1 children return FAILURE. Otherwise, it returns RUNNING.

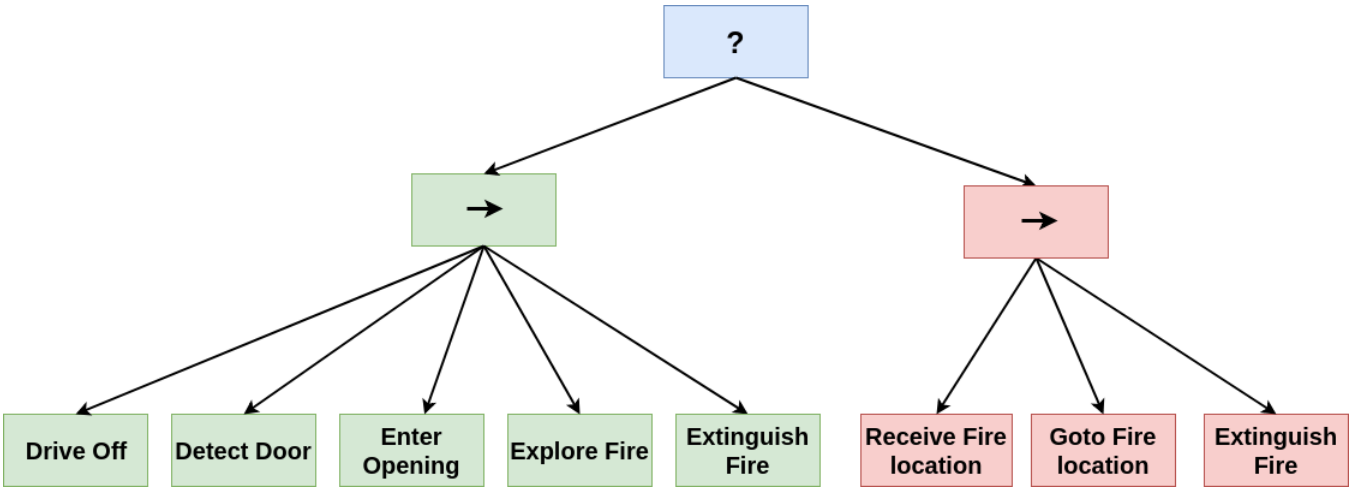
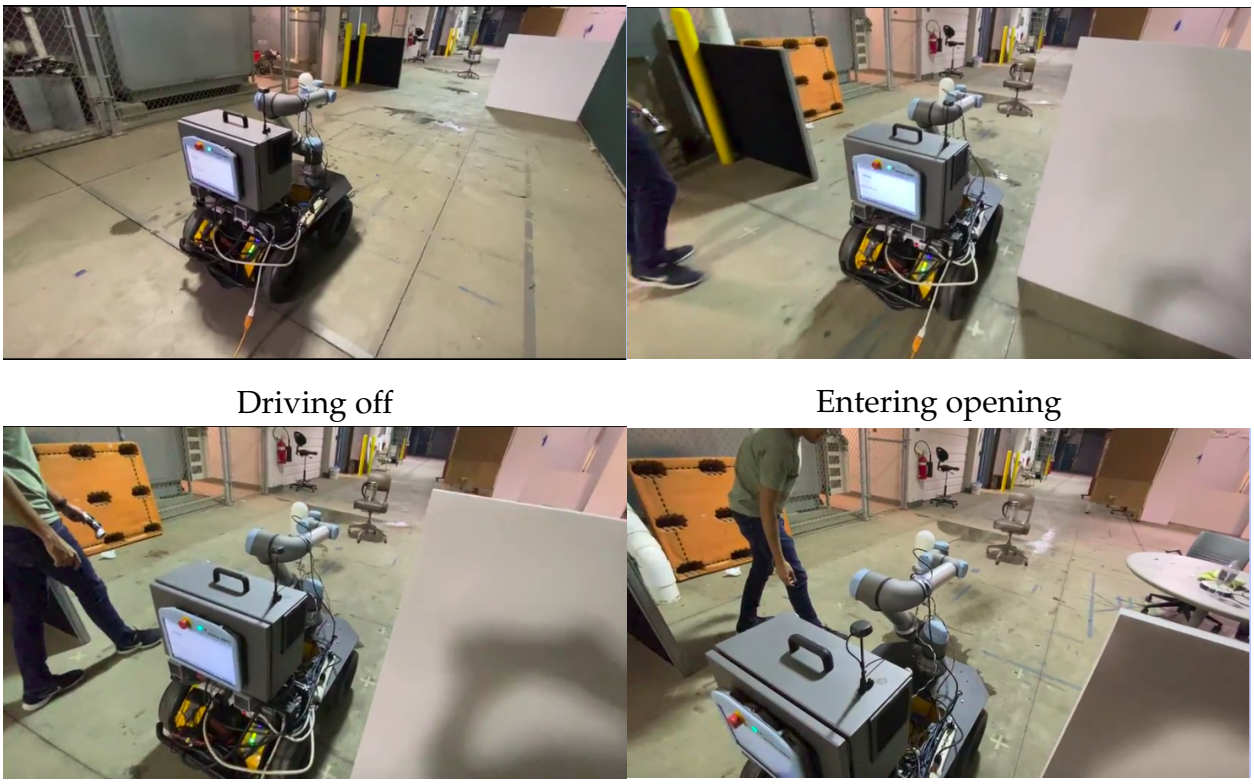


Figure 1: Behavior tree for UGV's mission

So, we have seven different actions in our behavior to perform the FVD mission. We performed a miniature version of FVD mission with four of these seven tasks (except explore, receive fire location & goto fire location). All four tasks/actions can be seen in Figure 2.



Driving off | Entering opening

Detecting Fire | Extinguishing Fire

Figure 2: Electronics for actuating water pump on Husky

Improved robustness of door detection:

Point cloud data from all the three front cameras are converted into the laser scan. This laser scan gives us reading from 0° to 180° in a resolution of 0.5°. We were using a simple line sweeping algorithm to detect rising and falling edge but our algorithm was robust enough to handle cases when there are few objects placed near the opening. So, we tried to implement the sliding window algorithm as we did it in window detection but it was not working straight away. All the values above 10.0m are marked as infinity during point cloud to laser scan conversion. But since there were infinite values in the laser scan, the sliding window algorithm will not algorithm (sum of left and right-hand side of the array will become infinite). So, all the laser scan values above 10.0 are saturated to 10.0m instead of infinity.
Then whenever there is a rising edge, it is marked as the starting point of the opening and falling edge as the endpoint of the opening. This (r,θ) pair is converted into (x,y) coordinates to get the centroid of the opening which will be the goal position for UGV.

Challenges:

Some of the challenges faced in the last two weeks are discussed below:

1. We need to fine-tune multiple parameters inside husky navigation packages and faced multiple issues due to inexperience with these packages.
2. We had to spend a good amount of time figuring out the mechanism to power the pump on the drone.
3. During our mission testing UAV crashed due to small bug in landing code and we spent a good amount of time salvaging the motors/electronics from the broken part.
4. Our Husky is shared with another team working on the DoE project. So, we had multiple scheduling conflicts and we had to plan proactively.
5. Since we are using multiple cameras, sensors, UR5e arm on Husky which are interfaced with Intel NUC. So, it is not possible to connect any other cameras/sensors on it. So, we are using Jetson TX2 for thermal processing and using multi-ros set up to communicate with NUC. In our earlier setup, Jetson was the slave and NUC was the master. So, Jetson was not able to receive any message from NUC (we need these for the behavior tree framework to work). Hence, we changed our interface to master-master interface (bi-directional) where both can listen to each other.
6. We tried different ways to fix the UGV wobbling issues where we even tried changing the positions of different sensors, batteries, and UR5 arm. We even tried outdoor wheels but nothing worked. Finally, we used duct tape to fix the wobbling issues.

Teamwork

We all helped each other in debugging various issues and brainstorming different approaches and algorithms.
Shubham & **Akshit** worked on tuning DWAPlanner parameters for the husky to get better performance while performing autonomous missions.
Akshit & **Parv** tried different strategies to overcome the power issues on the drone
Shubham worked on testing water tank/pump and all the electronics to control the extinguishing subsystem on UGV.
Steve helped all of us in setting up the logistics (tent on NSH-level B) and arena for testing UAV & AGV.
Shubham & **Akshit** brainstormed and tried different ways to reduce husky wobbling and finally wrapped duct-tape on the wheels.
Akshit & **Shubham** worked on testing, improving multi-ROS setup between NUC and Jetson.
Parv helped **Shubham** in understanding the behavior tree and helped him in debugging various issues during mission testing.
Parv & **Shubham** brainstormed together to improve the door detection on UGV.

Future plans

1. **Akshit** and **Shubham** will be work on developing the exploration strategy for fire detection
2. **Shubham** will be responsible for writing the ROS script to send/receive fire location between UAV and UGV for collaborative missions
3. **Shubham** & **Steve** will work on writing and testing missions for the Husky.
4. **Parv** & **Akshit** will be work on repairing the UAV.
5. **Parv** will work on integrating UAV global planner with the existing framework.
6. **Parv** will work on making a common map between UAV and UGV.