CMPE 565 AUTONOMOUS ROBOTS
Assignment 9: Subsumption Architecture

Deadline: 26 April 2018

In this assignment, we will implement a simple subsumption architecture in ROS. The orginal paper is here `http://people.csail.mit.edu/brooks/papers/AIM-864.pdf`. The main idea is to decompose the robot control problem into layers with different responsibilities. Therefore, every layer will deal with a certain behavior. It enables us to develop a robot control architecture which can be easily tested. Every layer can have its own representation of the world and produce actions accordingly. There is also priority of behaviors.

The objective of the robot is to find the red ball and going towards the ball. To achieve this behavior you are supposed to develop 3 different behaviors with the following priorities and tasks.

- **Level 0:** Obstacle Avoidance. When robot gets closer to an object, it will turn until there is no object in front of it. (use laser sensor data) (You will complete the behavior class)
- **Level 1:** Going to ball. When the robot sees the ball via its camera, it will go towards the ball. (use vision sensor, image_helper.cpp) (You will complete the behavior class)
- **Level 2:** Search for the ball. The robot will wander around to find the ball. Therefore, it will use the **incomplete map** to make efficient search. (you will use slam_gmapping and map_helper.cpp) (You will implement whole behavior)

We provide a basic subsumption architecture with two empty behaviors. You are supposed to write required functionality for given two behaviors and also add a new search behavior which extends **MyBehavior** class and overrides init and run methods like in the already implemented behaviors.

my_subsumption.cpp provides the implementation for ROS node. It subscribes to sensors and publishes required commands. It uses **GeneralSensorHelper** and **MyArbitrator** classes and behavior classes to implement the subsumption architecture.

GeneralSensorHelper is a helper class to wrap the required sensor data. This way we achieve to provide all sensor data with a class at every time step. It uses **MapHelper** and **ImageHelper** for specific sensor processing. You should write your own implemetation for **MapHelper::processMap** method according to your search behavior implementation. To understand how to process occupancy grid map, investigate isOccupied() method.

ImageHelper::processImage method is already implemented. It converts current image to opencv image and search image for the red pixels. If it finds, it sets **isBallKown**, **centerx**, and **centery** fields.

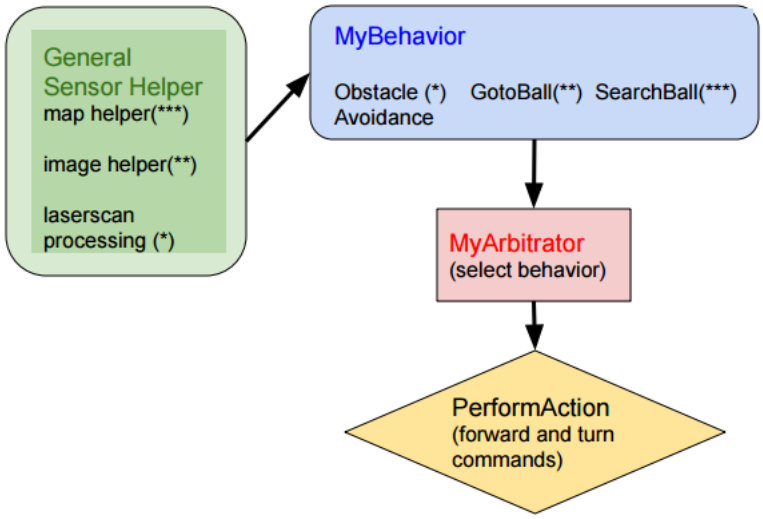

Figure 1: Code Architecture

You will complete the obstacle avoidance behaviour. For that, you need to also change/enhance GeneralSensorHelper::processLaserScan method to get the orientation of the obstacle.

It is important to understand that methods of **GeneralSensorHelper** class are not called directly, they are called by the subscribed ROS topics. Therefore, in these methods you should process the incoming sensor data, set certain fields accordingly.

MyArbitrator class is the core class of subsumption architecture. To add a behavior at certain level, we use **addBehavior** method. You should add behaviors in correct order since first added behavior is level 0, second added behavior is level 1, and so on. It checks the subsumption field of behavior classes starting from the level 0, and if subsumption field is set, it gets the action command of the that behavior and ignores other behaviors. The ROS node gets the action of arbitrator and publishes as the action command. Note that there is only forward and turn action commands considered for action.

You will use **MyBehavior** class for your "search for ball" behavior implementation. Once the node is created, init method is called. At every time step run method is called. Therefore, you should set subsumption field true to activate the behavior according to current sensor values, then generate forward and turn command. If lower level behaviours do not subsume it, these action commands will be used by the robot. Note that behaviors reach current sensor data through **generalSensorHelper**.

Before running any launch file complete these steps:

- run roscore
- run vrep simulator
- open assignment9.ttt scene file
- run the simulation
- run launch file

behaviortest.launch you can test your behaviors with this launch file. It basically runs the my_subsumption node. With these you can test all the behaviors except the searching for the ball.

findtheball.launch runs the nodes same as the behaviortest.launch, but also runs rviz and gmapping package.

1 Deliverables

- You should report the **average time**, your robot finds the ball for the current scene setting for 3 runs (If your implementation uses some random approach, then you can increase the run number to get a proper value).
- Let's think about a behavior where the user can control the robot through keyboard. If we integrate this behavior to the current architecture as the highest priority behavior, how will the overall behavior of the robot change? Please comment on briefly.
- You should provide a **video** showing the overall search behavior in simulation and on rviz. You should add annotation for cases where behaviors from different levels activated such as obstacle avoidance, or go to ball. We should clearly see that automatic behavior changes.