Ryu hyunwoo

#Overall Architecture

The overall Architecture resolves into 4 layers: Decision layer, Behavior layer, Perception layer and Actuator Layer.

Decision layer makes strategy based on perceptions and behavior layer concretizes its execution plan with actuator layer. Agents in perception layer interpret raw sensor data and yield useful informations. These informations are then sent to Blackboard.

Blackboard serves as a hub for robot informations. Every information gathers in blackboard and decision/behavior agents consult those informations that are stored in blackboard. Blackboard locates in behavior layer since many behaviors require quick response. Usually the total system spins at 10hz with ROS network. However, Behaviors and executors require different scheduling which makes them incompatible with ROS. Thus for necessity, blackboard object is instantiated in behavior node.

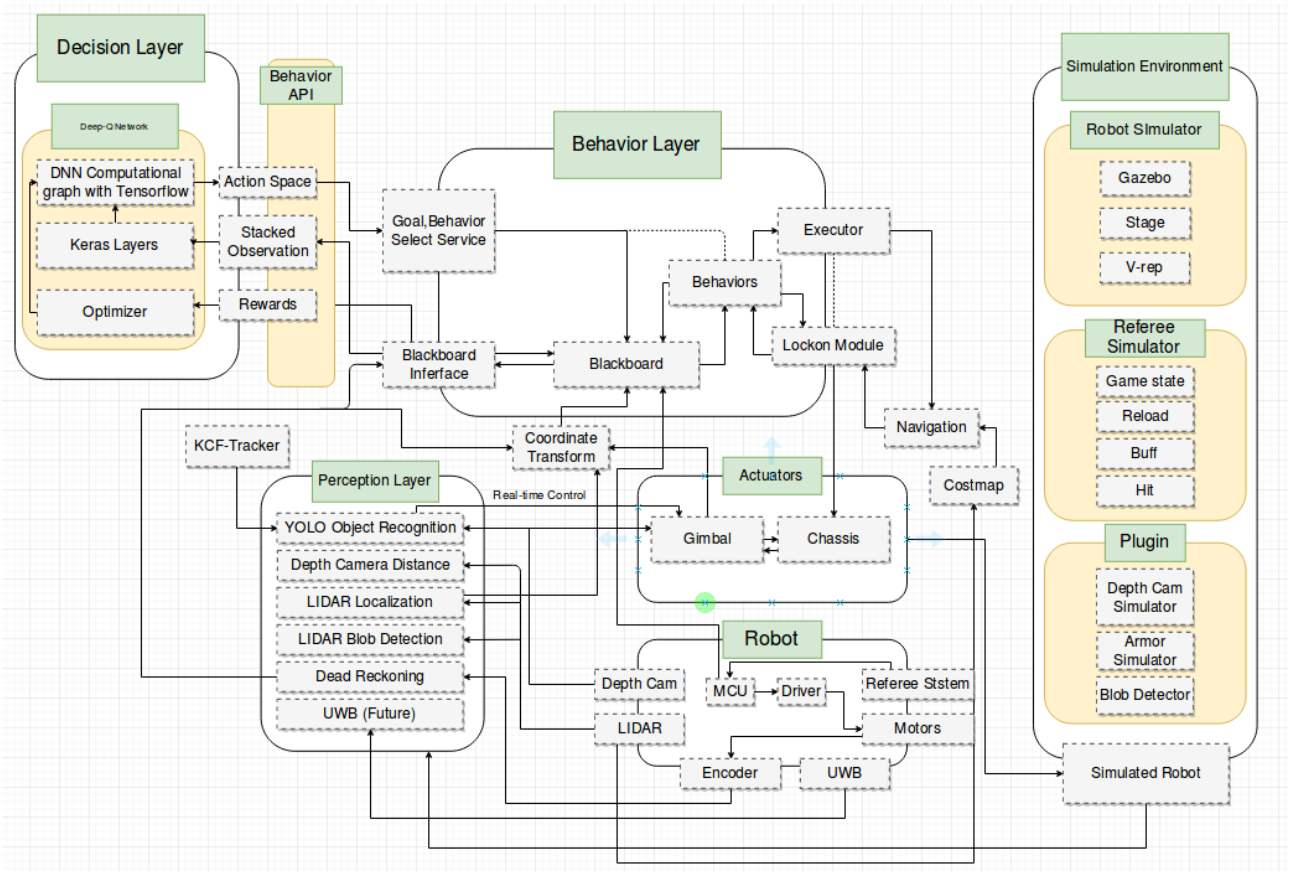
Decision layer gets processed information from blackboard with API. Only 3 functions are used for this communication : GetInfo, SetBehavior, SetGoal. Behavior and following goal parameter decided by decision layer is concretized in behavior layer. For example, if decision layer requests search behavior to coordinate (4,5), actual plans are made by behaviors. They plan the path using path planning modules, decides which angle to look at while moving, turn to the direction when hit, whether or not to lock on enemy, and so on.

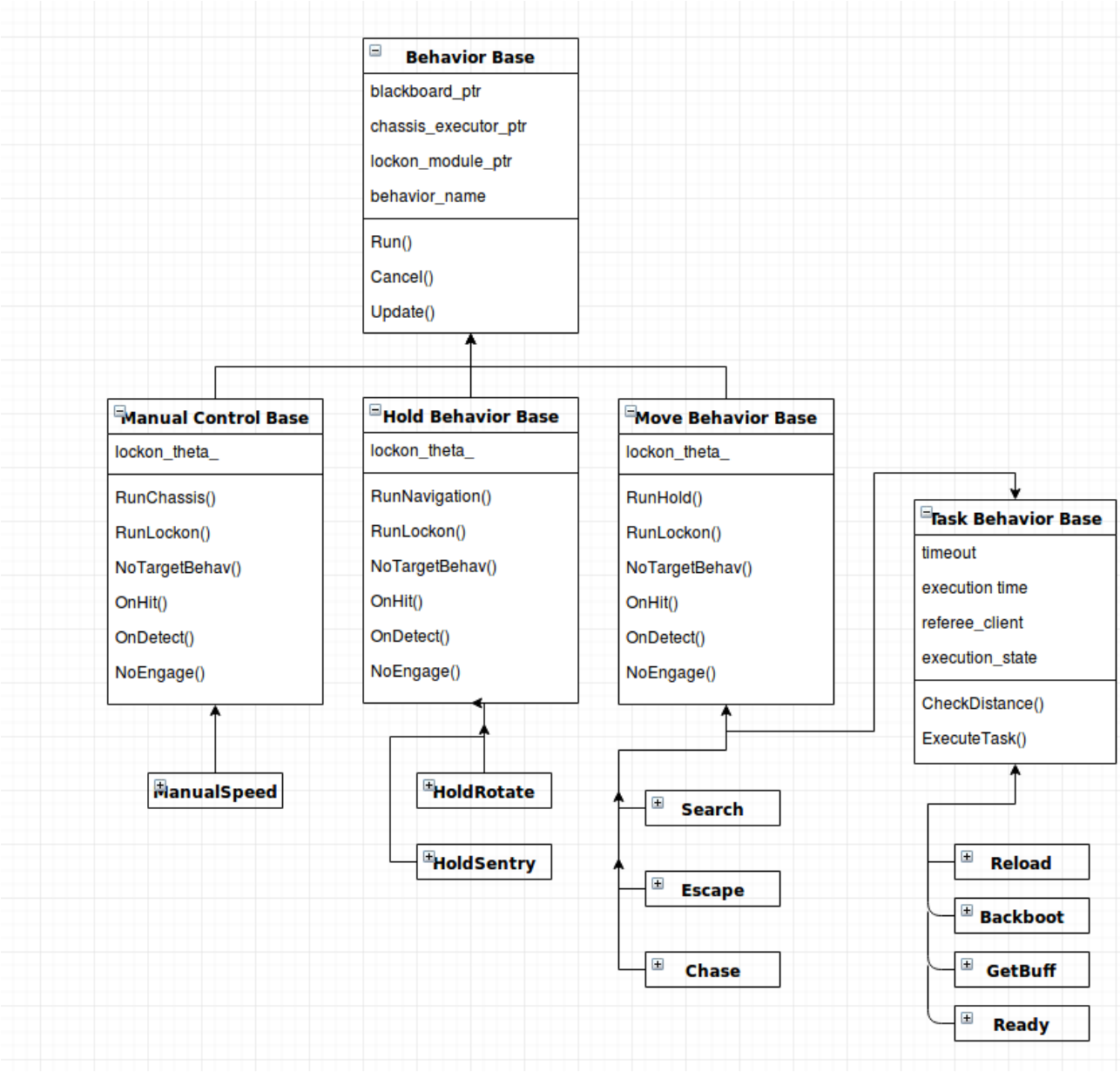
This architecture can significantly reduces burden for decision agent. Since every data are processed and interpreted by perception layer and detailed executions are performed by behavior, learnings can be done very easily. If we use deep reinforcement learning as a decision agent, for example, first there is no need to use big convolutional network to process raw camera data. This significantly reduces number of parameters to be optimized.

Second, action space is very small and abstract. The robot doesn't have to start from scratch. Robots with actionspace of simple x,y velocity need to learn first how to make path to enemy without collision. If rewards are given only when it hits enemy, the reward will be significantly dilluted. If one chose to give advantage for other factors, like move distance, this will make model significantly biased. On the other hand, our robot can reach enemy even at its first attempt as it uses navigation goal as actionspace. This will significantly boost learning.

Lastly, abstract actionspace helps the robot avoid being biased and overfitted to learning environments. As a result, simulated reinforcement learning parameters are directly applied to real robots. Due to such superiority in architecture, our team succeeded in using simulation trained model in real robot without any modification.

Below is the architecture of the overall system.


Decision Layer
Behavior API
Deep-Q Network
DNN Computational graph with Tensorflow
Keras Layers
Optimizer
Action Space
Stacked Observation
Rewards
Behavior Layer
Goal, Behavior Select Service
Executor
Behaviors
Lockon Module
Blackboard Inferface
Blackboard
Navigation
KCF-Tracker
Coordinate Transform
Perception Layer
Real-time Control
Actuators
Costmap
YOLO Object Recognition
Depth Camera Distance
LIDAR Localization
LIDAR Blob Detection
Dead Reckoning
UWB (Future)
Gimbal
Chassis
Robot
Depth Cam
MCU
Driver
Referee Ststem
LIDAR
Motors
Encoder
UWB
Simulation Environment
Robot SImulator
Gazebo
Stage
V-rep
Referee Simulator
Game state
Reload
Buff
Hit
Plugin
Depth Cam Simulator
Armor Simulator
Blob Detector
Simulated Robot



 # Localization

In our algorithm stochastic locating of robot with LIDAR is performed with particle filter and weighted Monte Carlo sampling. Bayesian inference and motion update is performed for each single steps in Markov process to stochastically evaluate individual particles. Each particles are weight sampled, thus converges quickly and unbiased. Number of particles are adaptively reduced or increased with respect to its deviations. Other sources besides LIDAR and dead reckoning, such as UWB can be integrated to estimation but not implemented yet in our project. Open source ROS AMCL package is used with parameter adjustment for holonomic motions

 # Motion Planning

Motion Planning Consists of Two parts: Global planner and local planner.

Global Planner

Our global Planner uses Dijkstra's algorithm in path findings. Although A* algorithm is faster than Dijkstra's, A* doesn't always yields optimal solution depending on which heuristic function used. In our motion planning, not only the distance but the weighted distance from

obstacles are also reflected to the heuristic functions. Also, as obstacle avoidance is much more emphasized than distance-optimal path, using A* ,which is somewhat lazy in avoiding obstacles doesn't seem to be a good option for us. Thus our robot planner used Dijkstra's as it is expected to navigate through complex costmaps. Our planner uses ROS navfn package.

Local Planner

Local Planner makes velocity plan using our Chassis Lock-on algorithm to exploit the advantage of holonomic motions. Our Chassis Lock-on algorithm supports any transversal trajectory with arbitrary angular velocity, thus enables our 'Lock-on' feature. 'Lock-on' feature means that the robot can make any movement in x and y axis while its yaw is fixed to certain target(usually the target enemy). As the robot's gimbal head angle is restriced to 180 degrees, this 'Lock-on' feature is essential for robot to shoot while performing any kind of movement. For example, the robot doesn't need to show its back (which is a dead-angle for its gun) or weak area to enemy in order to run away from that area.

Our Lock-on Alogorithm uses Dynamic Window Approach to make trajectories, but does not directly use the speed produced from it. It rather approximates the trajectory of DWA in quadratic scale with arbitrary angular velocity, thus enables our lock-on feature. DWA in three velocity space is only used to create and evaluate planar trajectory with certain curvature.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{(dtw')^2 + 4} \begin{pmatrix} dt^2 w w' + 4 & 2 * dt(w' - w) \\ -2 * dt(w' - w) & dt^2 w w' + 4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Our quadratic approximation is superior to mere linear approximation as it takes account of the angular velocity which acts as a accelerative force to transversal speed thus curved trajectory can be made within each steps. For example, when linearly approximated control loop of 10hz (0.1sec) with very fast angular velocity such as 5 rad/sec would have an error about the scale of timestep*angular_velocity which is about 0.5 m/s which is 25% percent of robot's maximum speed. Thus linear approximation for holonomic motion requires much faster control loop to safely follow curved trajectory which is infeasible. On the other hand, our algorithm can follow curved trajectory precisely and safely in high speed , slow control rate environments. Empirically, 10 hz control loop is just enough to drive safely while exploiting the robots maximum speed (2m/s). The approximated trajectory is undistinguishable from the original.

Our algorithm is superior to solely using DWA as ours can perfectly exploit the advantage of holonomic robots. Our algorithm is superior to TEB in that ours is much more fast and lightweight. TEB is an extremely heavy algorithm so it is not suitable for low-performance computing systems. TEB is especially bad when many obstacles exist. Time to time, the CPU and memory usage for calculation doubles, triples, and even quadraples, and eventually slows down the total robot system loop and enemy recognition or cause memory shortage. Although TEB yields very flexible and optimal trajectory, it seemed excessive for our system.

Open source ROS DWA planner package is used for trajectory generation.

 # Automatic Supply

Our automatic supply behavior starts by navigating to the reload zone and pre-align while moving. When it arrives to the zone, the robot fine-tunes its pose to align to the supplier. Currently, our navigation algorithm works just fine for alignment. However, PID control with LIDAR distance will be integrated to our system.