Mobile Robot Architecture for EECS 476/376, Spring 2011

Wyatt Newman

This document describes the software architecture that you should follow in constructing your mobile-robot code. The architecture is described in terms of nodes (modules), the respective interfaces (subscriptions and publications), and the intended behaviors.

Goal Publisher Node:

Behavior: sets (and updates, if necessary) goal poses (not to be confused with continuously-streaming incremental desired poses). Goal pose might be hard coded and set once only, or goal pose might be updated frequently with a sequence of sub-goals. Goal might be a vector of equivalent, alternative goals, allowing the planner to choose among options.

Publishes: poseGoal, consisting of: xGoal,yGoal, psiGoal (or a list of alternative goals)

PSO Node:

Subscribes to: low-level sensor information from cRIO and from GPS

Behavior: create a best estimate of robot's position, orientation and actual speed/spin. (At present, this node has been created for you).

Publishes: poseActual--estimate of robot's actual position, actual orientation and actual speed/spin,

LIDAR Driver Node:

Acquires raw data from LIDAR and publishes it at max rate (LIDAR refresh rate, nom 75Hz)

Publishes: LIDARscan

LIDAR-Based Mapper Node:

Subscribes to:

- LIDARscan
- poseActual

Behavior: create/continuously update a C-space map in world coordinates using LIDAR scan data and corresponding robot poses

Publishes: LIDAR-based C-space map;

Note: avoid subscribing to C-space maps over WiFi (communication bottleneck for huge packets)

Path Planner Node:
subscribes to:

- LIDAR-based C-space map
- Vision-based C-space map
- Sonar-based C-space map
- poseGoal (destination pose)
- poseDes: Current desired pose (x_des, y_des, psi_des)

Behavior: Using current desired pose as a start point and published goal point(s), derives a collision-free path from start to goal in the current map(s). Expresses the resulting path as a sequence of path segments, per the path description format. If a path to a goal cannot be found, the planner provides a partial solution, e.g. invoking a bug algorithm.

The planner accepts C-space map updates and checks if the existing solution is still valid (by sampling points along the path and testing the corresponding C-space pixels for occupancy). If a current plan is found to be invalidated by a revised C-space map, the planner performs a new path-planning solution.

The planner checks if the goal location has changed, and if so, replans a path to the new goal.

Publishes: pathList--a currently-valid plan (to the extent possible), using a defined path datastructure. Path data structure includes: segNum, segLen, segType, referencePt, initTanAng, curvature, max speed/spin limits, max acceleration limits.

Desired Path Crawler Node:

Subscribes to:

- SpeedNominal
- pathList

Behavior: Keeps track of (and increments) segNumber (current segment index) and lsegDes (scheduled path-length progress along current segment). Uses speedNominal and delta-time to update lsegDes. Uses segment descriptor and lsegDes to compute desired values xDes, yDes, psiDes, rhoDes. Should update at minimum 10Hz. (Faster is better).

Publishes: CrawlerDesState, consisting of lsegDes, xDes, yDes, psiDes, rhoDes, segType, segNumber

Speed Profiler Node:

Subscribes to:

- CrawlerDesState

- pathList
- CspaceMap(s)

Behavior: Starting from the current scheduled point along the pathList, test path samples into the future for at least the braking distance (braking distance plus tolerance). If the path is clear, ramp up the value speedNominal, if below the speed limit, or ramp down speedNominal if approaching a stopping point or lower speed limit segment. If the sampled pathList reveals obstruction by a C-space obstacle, ramp speed down to schedule braking to a halt before a collision.

Publishes: speedNominal (and copies over elements of CrawlerDesState to bundle together with speedNominal—thus can re-use message type published by path-crawler node).

Steering Node:

subscribes to:

- poseActual
- CrawlerDesState
- speedNominal

Behavior: given desired and actual states, computes the transformed error variables of lateral offset error, heading error and following error. For type-3 path segments, interprets path length and following error in terms of radians instead of meters. Based on the following error, computes a correction to the speedNominal (to null out the following error) resulting in a speed command. Using the speedNominal, the heading and lateral-offset errors, and rhoDes, computes an appropriate spin-rate command. This computation should be reiterated relatively quickly (e.g. 50Hz).

Publishes: speed and spin commands (used by the PID controller to control wheel motions)

Camera Driver Node:

Grabs snapshots from camera(s)

publishes images. (May have multiple camera drivers and multiple image messages)

Vision Mapper Node N:
subscribes to:

- camera image (may be on of multiple published sources for multiple cameras)
- actualPose

Behavior: Specialized vision mapper node looks for specific features (e.g. orange objects, white lines, green flags, ...). Computes world coordinates of found objects (assuming a given height, e.g. ground level). Depending on specifications, may provide C-space transformed map. Updates global map with new images. Like LIDAR-based mapper, may require averaging, low-pass spatial filtering, forgetting, map clearing, etc. Mapping requires use of camera calibration information (both intrinsic and extrinsic), dewarping, projection onto plan view, and accumulation within a global map.

Publishes: visionMapN (an occupancy-map array in world coordinates based on observations of the specified visual features)

Sonar Driver Node N: acquires raw data from sonar sensor N and publishes the sonar data (a single "ping" radius). Repeats at max practical rate (e.g. sonar ping repetition frequency). Optionally, a single driver might handle multiple sonar sensors and report out a vector of ping radii, sonarPings.

Publishes: sonarPingN or sonarPings

Sonar Mapper Node: (note—this description of how to access sonar data will likely change)

subscribes to:

- sonarPings
- actualPose

Behavior: interprets sonar pings (using known poses and known sonar mounting transforms) to populate a sonar-based obstacle map in world coordinates, suitable for overlay with LIDAR and camera-based mapping.

Publishes: sonarMap