[Instruction] To get the credits for the challenge each group should publish their source code on GitHub together with a description of the project. Will be linked from the WASP web page.

Overview

Problem Outline

The goal of this project is to develop a prototype autonomous solution for a Search-and-Rescue (SAR) mission. The autonomous agents are ground robots and the drones that deliver supply packages in an unmapped environment to the persons in need after, e.g., an earthquake. This project is submitted towards the WASP Autonomous Systems Challenge 2016 [1].

System Setup

This project is developed using ROS in a Linux environment and makes use of several open-source ROS packages for planning, control, localization, and perception. We use a single ground robot and a single drone that are specified below. The environment is setup in a lab area with uniform floors and is completely enclosed by vertical walls/doors. Any glass boundaries are covered to ensure robust mapping by the range sensors on the ground robot. There are multiple randomly located obstacles such as tables and cardboard boxes of varying sizes scattered within the environment. The persons in need are represented by clipart printed on paper and taped at random locations on the floor. Each person is marked with a unique ARTag located near the clipart.

Hardware:
Turtlebot 2 mounted with a LIDAR system (RPLIDAR) and a 3D Sensor (Asus Xtion Pro)
ARDrone 2.0 (off the shelf)

Operation Strategy

The environment is assumed to be unknown at the start, i.e., the environment map and the locations of obstacles/persons within the environment are unknown. The operation of the system is outlined below, categorized according to common aspects of an autonomous system:

Planning: The SAR mission comprises two phases, an “exploration phase” and a “delivery phase”. Immediately after startup, a plan is generated and executed for exploring the environment. During this phase, the ground robot drives through multiple exploration waypoints (loaded from a file) and a map of the environment is generated using the LIDAR. During this phase, the location of persons is detected based on the ARTags and stored within a database. At the end of the exploration phase, the ground robot returns to the loading area. The delivery phase is then initiated, wherein a new plan is generated and executed to deliver the supplies to the persons. In this phase, the drone carries a supply package between the package area and loading area, and the ground robot delivers this package further to a person. The path planning for turtlebot is achieved by its open-source move_base package.

Control: The ground robot motion is uses in-built Turtlebot controllers. The drone motion uses simple PID controllers developed within this project.

Sensing: The ground robot fuses IMU data and LIDAR for Simultaneous Localization and Mapping (SLAM). The drone SLAM is achieved by fusing IMU data and monocular vision based on rich visual cues (high-contrast newspaper clippings) and ARTags.

Perception: During the exploration phase, the location of persons in need are identified by scanning for ARTags in the image stream from the 3D sensor on the ground robot.

Interaction and Collaboration: The drone carries a supply package from the package area to the loading area, and loads it onto the turtlebot. This system does not incorporate human intervention during the system operation, although the sensor outputs can be visualized on a RViz interface.

System Description

Planning

Packages: https://github.com/vidits-kth/WASP-AS-KTH3/sar_autonomy/sar_planning
https://github.com/vidits-kth/WASP-AS-KTH3/sar_autonomy/sar_exploration

The planning system for the exploration and delivery phases makes use of KCL ROSPlan [2][3]. ROSPlan requires the domain description in Planning Domain Definition Language (PDDL) format. The domain description in this project requires STRIPS and supports durative actions. This domain description is used to generate problem and plan files at run time. ROSPlan provides a “Knowledge Base (KB)” to keep track of the system model obtained from the domain file, instances of the defined types, and system state. The types defined in this project are *turtlebot, drone, waypoint, person, crate* and *content*. The durative actions defined are *pick-crate, load-crate, deliver-crate* and *drive-to.* The ROSPlan planning system is used to:

1. Fetch information from the KB and generate a PDDL problem file.
2. Call a planner and generate the plan.
3. Parse the plan and dispatch actions via ROS messages.

Initialization: At initialization, the ROSPlan planning system loads the domain into the KB from the domain PDDL file. Further, the initial system state is programmatically added to the KB such as the ground robot, the drone, and the supply packages. The initial locations of the ground robot and the drone are assumed to be the loading area (“origin” in the *map* frame) and the package area respectively.

System State: The current state of ground robot and the drone, and the known scenario elements (e.g., supply packages) are updated in the knowledge base whenever any appropriate

action gets executed. The scenario elements that are not known at the beginning of the SAR operation (e.g., persons in need and their locations) are added dynamically based on relevant sensor inputs during the exploration phase.

Goal States and Plan Generation: Immediately after initialization, a set of exploration waypoints are added programmatically as goal states for the exploration phase. The planner then gets called to generate a plan for driving the ground robot through each of these exploration waypoints and subsequently return to the loading area. The plan is generated by the planner bundled within ROSPlan, a Forward-chaining Partial Order Planner, POPF [4].

At the end of exploration phase, new goal states are added dynamically to deliver supplies to all persons in need discovered during the exploration phase. The planner gets called a second time to generate a delivery plan which involves (1) moving the supply packages from package area to loading area using the drone, (2) loading the package onto the ground robot, (3) driving the ground robot to a person, (4) delivering the package, and (5) driving the ground robot to the loading area. This delivery operation is repeated until supplies have been delivered to all persons in need.

ROSPlan parses the plan generated in each phase, and generates a ROS messages sequentially for the action to be dispatched at each step. In addition to the PDDL description, there is also other system-related information to be maintained, such as the coordinates of the ground robot and persons in map frame. This project uses ROS mongoDB database for storing such information.

The ROSPlan planning system can handle replanning attempts upon planning or action failure, or plan invalidation. The number of replanning attempts, and a number of other parameters are configurable. The locations of the initial waypoints and persons can be visualized in RViz by subscribing to the topic */kcl_rosplan/viz/waypoints/.*

Turtlebot Localization

Package: https://github.com/vidits-kth/WASP-AS-KTH3/sar_autonomy/sar_gmapping
This project uses LIDAR-based SLAM for mapping the environment during the exploration phase using the ground robot, and for localizing the ground robot during delivery phase. The *Turtlebot Gmapping* package is used for SLAM and is configured to work with the *laser* messages generated by the *RPLidar_ROS* package. A few configuration tweaks have been made to the original *RPLIDAR_ROS* configuration for improved robustness, as discussed in [5].

ARTag detection

Package: https://github.com/vidits-kth/WASP-AS-KTH3/sar_autonomy/sar_apriltags
As discussed above, all persons in need are marked with a unique ARTag. The ARTag is sensed by the ground robot using the camera feed from the onboard Asus Xtion Pro camera.

During the exploration phase the ARTag detection node is running and it subscribes to raw camera feed, */camera/rgb/image_raw*. Since the turtlebot uses LIDAR for SLAM, the coordinates of the ARTag are transformed with respect to the *laser* frame to locations in the *map* frame.

Note: The ARTag detection node is running during the entire SAR operation (exploration + delivery), but the delivery plan is only generated once immediately after the exploration phase. As such, any ARTags detected during the delivery phase are ignored.

ARDrone Driver

The ARDrone Autonomy package is used to interface the drone with ROS [6].

ARDrone Control

Package: https://github.com/vidits-kth/WASP-AS-KTH3/sar_autonomy/sar_ardrone_control
This projects uses a modified version of the keyboard controller from Mike Hamer's Ardrone Tutorials package [7], that includes the control algorithms that allow the drone to go to a reference pose in the coordinate system. A state feedback controller with feedback from position and horizontal speed is used for position and a P-controller for yaw. Keyboard input from the user overrides the automatic controller.

ARDrone Localization

The drone localizes itself based on monocular visual slam from ORB-SLAM2 [8], using its downward facing camera. It uses ARTag detection at its origin in order to calculate the transform between its local frame and the system-wide *map* frame generated by the ground robot.

ARDrone Motion Planner

Package: https://github.com/vidits-kth/WASP-AS-KTH3/sar_autonomy/sar_ardrone_motion
The motion planning node translates action commands from the drone action server into time dependent reference poses for the controller.

ARDrone Action Server

Package: https://github.com/vidits-kth/WASP-AS-KTH3/sar_autonomy/sar_drone_interface
The drone action server is used as an interface between the ROSPlan planning system and the motion planner node.
Example, *Load* command:

1. The action server sends the liftoff command to the drone driver
2. Once the drone has obtained good localization in the air (having picked up a virtual box of supplies), the action server sends the command to go to the turtlebot location, to the motion planning node. The reference pose gradually moves to the turtlebot location, and the controller causes the drone to follow the reference pose.

3. At the turtlebot location, the action server sends a quick land and liftoff command sequence to the driver, to indicate a package delivery.
4. Go back and land at origin, same as (2)

For example when the *load* command is issued by the planner, the action server first sends the liftoff command to the drone driver, once it has obtained good localization (having picked up a virtual box of supplies), the action server sends the command to go to the turtlebot location, to the motion planning node.

References

[1] http://wasp-sweden.org/custom/uploads/2016/11/Challenge-AutonomousSystem-2016.pdf
[2] http://kcl-planning.github.io/ROSPlan/
[3] Cashmore M., Fox M., Long D., Magazenni D., Ridder B., Carrera A., Palomeras N., Hurtos N., and Carreras M., *ROSPlan: Planning in the Robot Operating System*, International Conference on Automated Planning and Scheduling, North America, apr. 2015.
[4] https://github.com/LCAS/popf
[5] http://geduino.blogspot.se/2015/04/gmapping-and-rplidar.html
[6] https://github.com/AutonomyLab/ardrone_autonomy
[7] https://github.com/mikehamer/ardrone_tutorials
[8] https://github.com/raulmur/ORB_SLAM2