Development Resources

Overall Design of the Robot Management Application

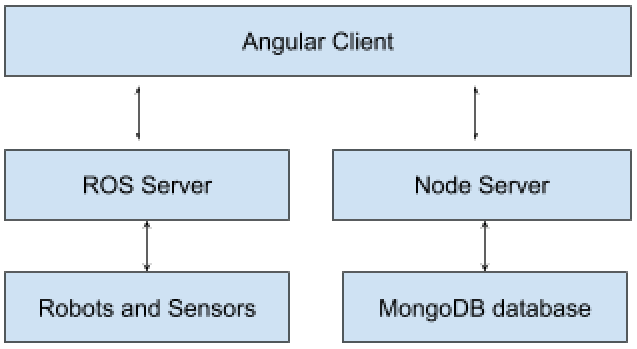
The application consists of three main components:

1. Angular Client
2. Node Server
3. ROS Server

The Angular client is written in Javascript and creates the user interface .

The Node server is written in Javascript and handles the user management system and stores user information in a MongoDB database.

The ROS server runs on the robot and facilitates communication between the various robot components and control software.



Angular Development

The overall structure of the Angular client is described in the `angular_app_structure.md` file.

To build additional functionality for the Angular client, you can find documentation on how to work with the Angular framework at https://angular.io/docs.

User Management System (Node.js) Development

To learn more about the user management system, refer to the `LoginSystemDesign.pdf` document.
To add additional functionality to the node server, you can find documentation on Node.js at https://nodejs.org/en/docs/.

ROS Server Development

To build additional ROS functionality, you can find documentation about ROS here: http://wiki.ros.org/

Package and Library List

The robot management application utilizes a number of existing ROS and Javascript packages to provide the visualization and robot control functionality in the client application. The following is a list of these packages and links to learn more about them.

ROS Packages:

The following ROS packages are used to connect the web application to the ROS server.

rosbridge_server

- Wiki: http://wiki.ros.org/rosbridge_server
- This creates a server that will allow the client to connect to ROS through a websocket connection. By using the roslibjs library, JSON messages can be sent from the client side web application to the rosbridge server running on the robot.

Kinect2_Bridge

- Documentation: https://github.com/code-iai/iai_kinect2/tree/master/kinect2_bridge
- This is a set of Kinect camera drivers that will allow ROS to access the video stream from the kinect.

robot_pose_publisher

- Wiki: http://wiki.ros.org/robot_pose_publisher
- This package publishes the robot's current pose so that nav2djs can display the robot's location.

screengrab_ros

- Wiki: http://wiki.ros.org/screengrab_ros
- This package publishes a portion of the screen on the topic `/image`.
- If you want to stream image from something like gazebo or rviz, this package can be utilized.

web_video_server

- Wiki: http://wiki.ros.org/web_video_server
- The web video server encodes an image topic on ROS and makes it accessible via the web.

    - For the project, it is being used to stream the Kinect video output and can also be used with the screengrab_ros package to display gazebo or other desktop programs.

Rtabmap_ros

- Wiki: http://wiki.ros.org/rtabmap_ros
- The rtabmap package is a SLAM (simultaneous localization and mapping) solution that utilizes the Kinect camera's information to produce maps.

Javascript Libraries used by Angular Client:

roslibjs

- Wiki: http://wiki.ros.org/roslibjs
- Documentation: http://robotwebtools.org/jsdoc/roslibjs/current/
- Roslibjs is the javascript library that is used by the Angular client to connect to ROS.
    - It can be used to publish and subscribe to ROS topics and call ROS services or set ROS parameters.

ros2djs

- Wiki: http://wiki.ros.org/ros2djs
- Documentation: http://robotwebtools.org/jsdoc/ros2djs/current/
- This javascript library is used by the nav2djs library to render 2D graphics.

nav2djs

- Wiki: http://wiki.ros.org/nav2djs
- Documentation: http://robotwebtools.org/jsdoc/nav2djs/current/
- Nav2djs creates the viewer for a map. A double click on the map sends a navigation goal to that location.

Library Dependencies:

- EaselJS
- EventEmitter2