UNIVERSITY OF BURGUNDY

ROBOTICS PROJECT

Explorator

Devesh Adlakha

January 6, 2016

Abstract

Explorator was developed as part of the Robotics project module in the third semester of study in the Masters in Computer Vision program at the University of Burgundy, France. The project requirement was an autonomous system integrating patrol (navigation and localization) and computer vision functionalities on a Turtlebot 2[1] platform running the ROS Hydro Medusa[2] distribution. Explorator is modeled after applications in exploration, search and rescue, and disaster operations, which form an active area of research in autonomous and semi-autonomous robotics. The task capabilities include: autonomous SLAM, global localization, surveillance, manual override, object recognition, hazard detection, RGB-D SLAM, and speech synthesis. The context of the work is first introduced, along with the envisioned scenario, providing an overview of the task capabilities and their relevance. Each task is then presented independently, followed by the integration to a working system. The challenges faced in the project development, and ideas for future work are finally discussed. The code is available on the following GitHub repository: https://github.com/Devesh99/Explorator. A video compilation of the results would be posted on the following YouTube channel: https://www.youtube.com/channel/UCZjeAf84p1_njtmVjmDFTaw.

CONTENTS

List of Figures

ABBREVIATIONS

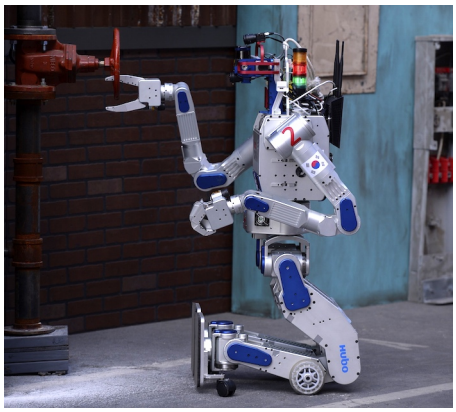
DARPA	Defense Advanced Research Projects Agency.
DRC	DARPA Robotics Challenge.
KAIST	Korea Advanced Institute of Science and Technology.
RGB-D	Red Green Blue-Depth.
ROS	Robot Operating System.
RTABMAP	Real Time Appearance Based Mapping.
SLAM	Simultaneous Localization and Mapping.
TTC	Time To Contact.

CHAPTER 1

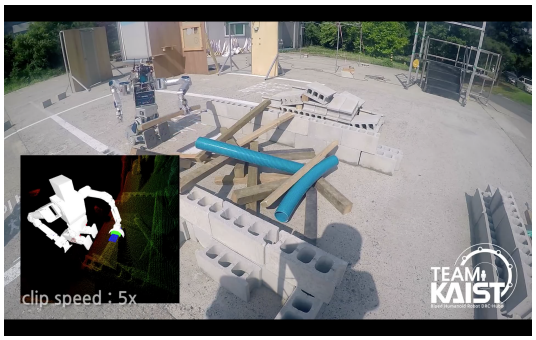
INTRODUCTION

Autonomous and semi-autonomous robots are suited for venturing in unknown or dangerous environments in applications such as exploration, search and rescue, and disaster operations. There is wide interest in the associated problems; the DARPA Robotics Challenge[3] aimed to accelerate progress in robotics to aid in dangerous environments, as a response to the nuclear disaster in Fukushima, Japan, in 2011. The recent DRC finals competition consisted of eight tasks relevant to such a scenario, including complex motion (driving, egress, movement through rubble, climbing stairs), object detection and manipulation (turning valves, tripping circuit breakers, using electric equipment), among others. The winning robot, HUBO from KAIST in South Korea, shown in figures 1.1a-1.1b, successfully completed the eight tasks, highlighting the benefits of its design combining both humanoid and ground robot forms.

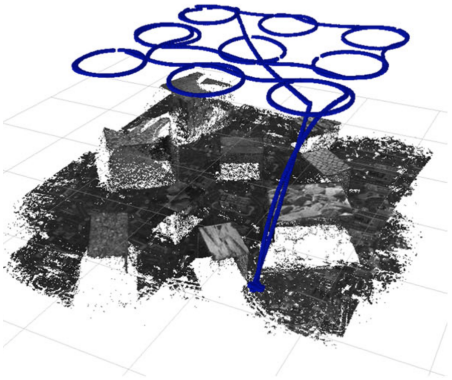
Research groups, such as the Robotics and Perception group at the University of Zurich, are also heavily focused on the challenges in autonomous robotics. Figure 1.1d shows the TEDx talk delivered by the head of this group, Davide Scaramuzza, demonstrating their micro flying robots in applications of search and rescue, inspection, and environment monitoring. These quad copters rely on vision-based methods for autonomous navigation, SLAM, and extend capabilities to multi-robot collaborations. Figure 1.1c depicts a resultant map along with the trajectory of the motion followed.



(a) HUBO operating a valve



(b) HUBO removing rubble



(c) SLAM with a micro flying robot

(d) Davide Scarramuzza: TEDxZurich

Figure 1.1: Autonomous robots

The motivation behind Explorator was to address some fundamental problems in Robotics and Vision through tasks relevant to such scenarios. The task capabilities are particularly inspired from project Mappotino[4], and include:

- Autonomous SLAM
- Global localization
- Surveillance
- Manual override
- Object recognition
- Hazard detection
- RGB-D SLAM
- Speech synthesis

The project was developed on the Turtlebot 2 platform with a mounted laser scanner(figure 1.2), running the `ROS Hydro Medusa` distribution. The environment was constrained to an assembled arena in the laboratory, shown in figure 1.3. The envisioned scenario incorporating the selected tasks is presented next.

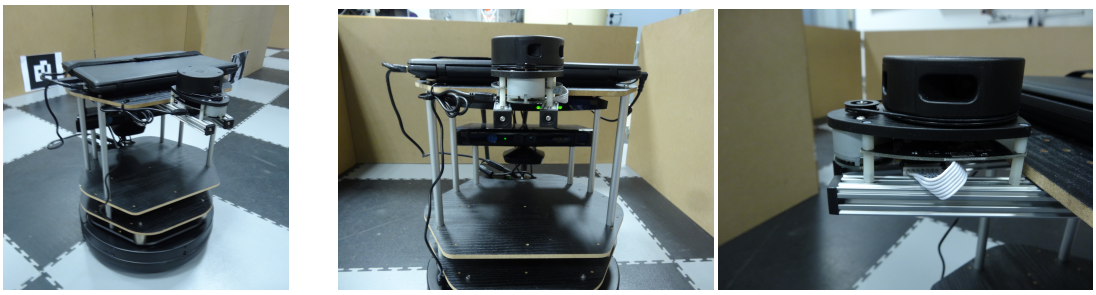


Figure 1.2: Platform: Turtlebot 2 with mounted RPLIDAR

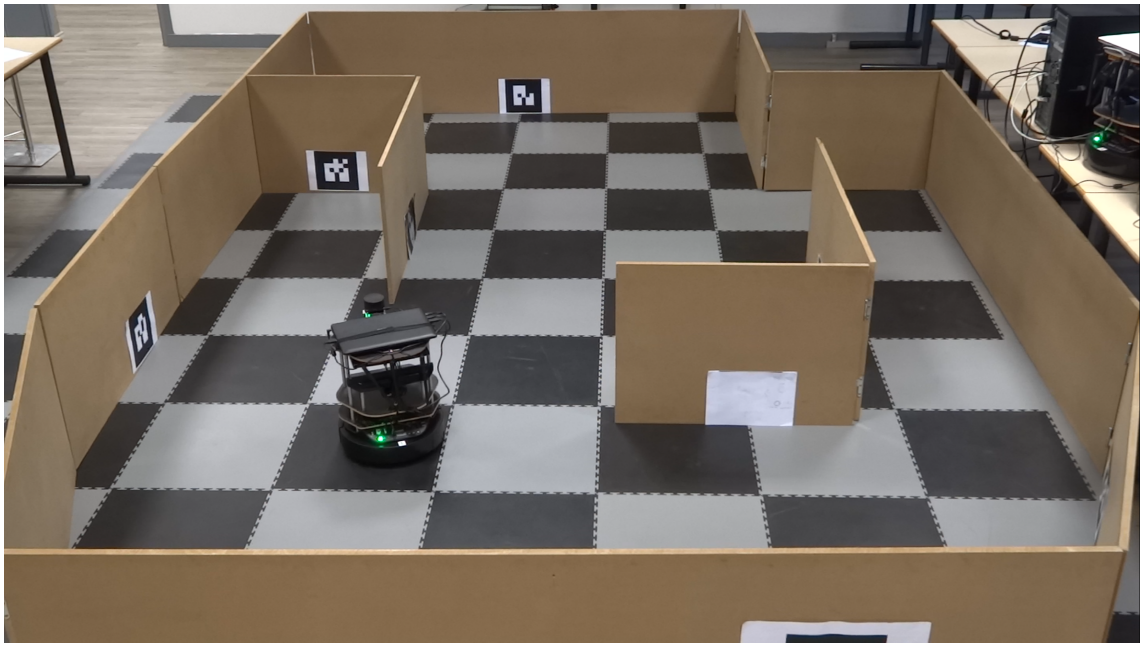


Figure 1.3: Arena: constrained environment for project development

1.1 SCENARIO

The envisioned scenario for this project consists of three stages:

1. Autonomous map building
2. Data collection
3. Analysis and processing

1. Autonomous map building
   A map provides vital information of an unknown environment. In a disaster scenario, communication with the robot may malfunction or be unreliable, preventing manual motion control of the robot. This necessitates autonomous motion for SLAM, or autonomous SLAM.
2. Data collection
   Data collection is the prime function of Explorator, and comprises the following tasks:

- Global localization
  A robot must be able to localize itself in its environment using the created map. While SLAM leads to both the map and localization within this map, for subsequent expeditions, or multi-robot systems, localization must commence from scratch. A modification of the environment, such as the addition of visual markers, is usually not possible, and localization must be achieved relying on sensor data and the created map. This is the global localization problem.
- Surveillance
  Surveillance includes patrolling the environment while collecting data of interest, such as camera images. Once localized, the robot may navigate around the map for this purpose.
- Manual override
  Manual motion control is necessary as a safety feature, as well as to assist in performing tasks. An overriding behavior is desirable, giving a higher priority to safety mechanisms and manual control over patrol.
- Object recognition
  Surveying the environment for objects of interest and maintaining a log of the identified objects as well as their position in the map provides useful insight into the environment.
- Hazard detection
  Hazards, such as fire, require an emergency routine to be followed. Such a routine could include stopping current tasks, and heading to a defined emergency exit.
- Speech synthesis
  Speech synthesis is useful in search and rescue, or disaster operations. In this project, it is used mostly for feedback for demonstration purposes.

3. Analysis and processing
   The collected data could be analyzed for scene understanding and adapting the robot behavior. For instance, the map could be augmented by adding the positions of the static recognized objects. Furthermore, a registered $3D$ reconstruction (RGB-D SLAM) of the scene better represents the environment for visualization as well as its understanding.

CHAPTER 2

FEATURES

2.1 AUTONOMOUS SLAM

SLAM is implemented and well-established in ROS, with the standard gmapping package, as well as other implementations. Conventionally, a map of an environment is created in the form of an occupancy grid using a laser scanner (or even a Kinect) along with teleoperation of the robot around the environment to map. The rationale behind autonomous slam is to simply replace the manual motion control with autonomous navigation around the environment. Figure 2.1 shows the two components of autonomous SLAM: a SLAM algorithm, such as gmapping, and an exploration-based navigation.

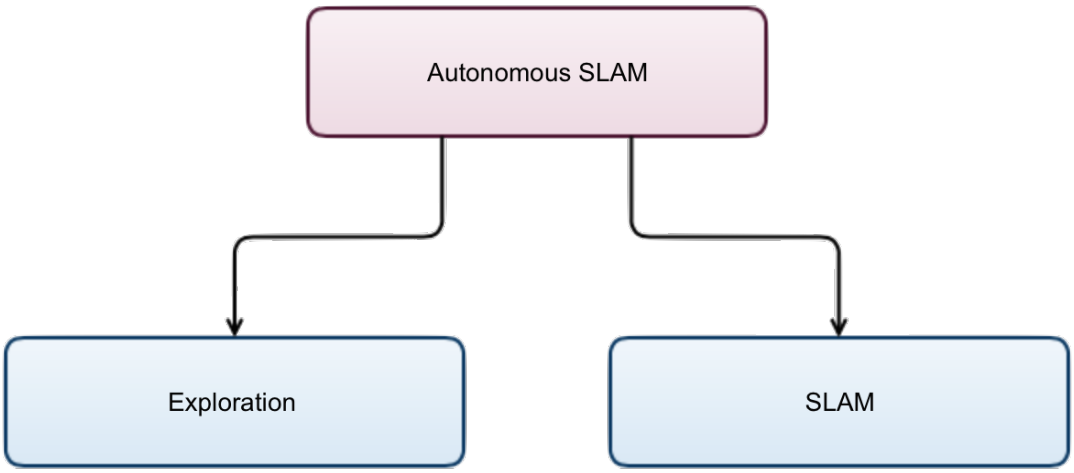
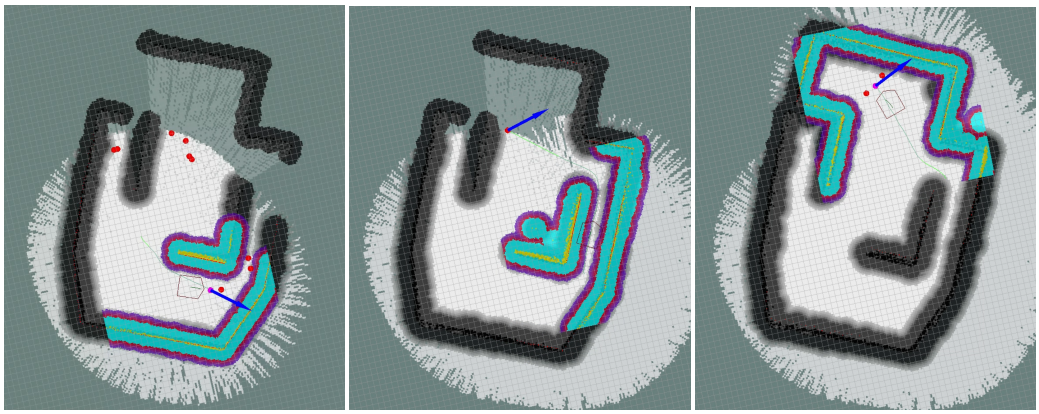


Figure 2.1: Autonomous SLAM: concept

The frontier_exploration[5] package implements the frontier exploration[6] algorithm, to explore an environment while looking for a closed boundary between known and unknown areas. Along with gmapping, they formed the components for autonomous SLAM in the scheme depicted in figure 2.1. The package provides a use case for a map source, such as from SLAM, whereby the cost map frames between exploration and SLAM match. A simple action client was implemented to specify an unbounded exploration, wherein exploration continues till a closed boundary is formed. The alternative is to specify a closed polygon to explore within, either graphically using RVIZ or through a simple action client. The working of unbounded frontier exploration is explained

below with reference to figure 2.2:

1. A number of frontiers are sampled, shown as red points, and based on a criteria one of them is selected as the goal to pursue. Here, the closest frontier is selected, depicted in purple. The exploration boundary is shown in black, along with its inflation radius. The local cost map is overlaid on this exploration boundary. The white areas represent explored areas, which are determined by the configured laser scan range.
2. As the robot moves, more areas are marked as explored, and the exploration boundary forms incrementally.
3. A closed boundary is eventually formed, and all areas explored within this space. This marks the end of the unbounded exploration.



(a) Frontiers: closest selected as navigation goal

(b) Boundary and explored areas incrementally form

(c) Exploration finished: all areas within boundary explored

Figure 2.2: Unbounded exploration

A single run of unbounded exploration was found to be insufficient in general to register scans and form the complete map. As such, the unbounded exploration is repeated for an empirically determined number of trials (4) in creating the map autonomously. Figure 2.3 shows the restart of the unbounded exploration, following the first pass of figure 2.2.

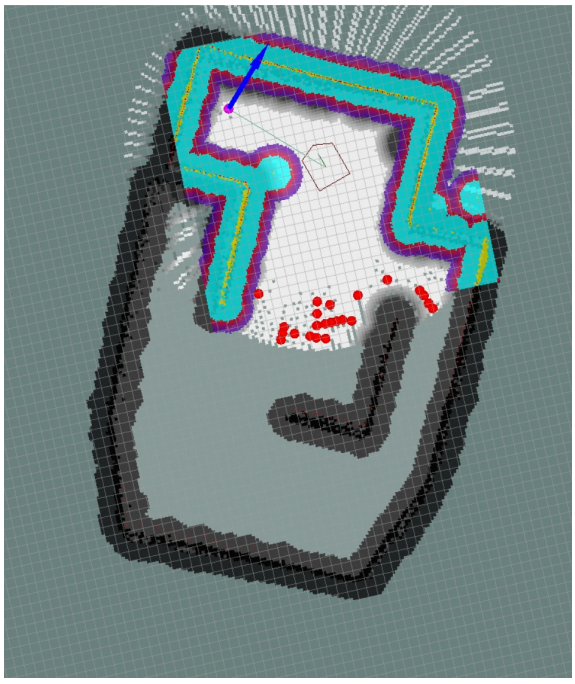
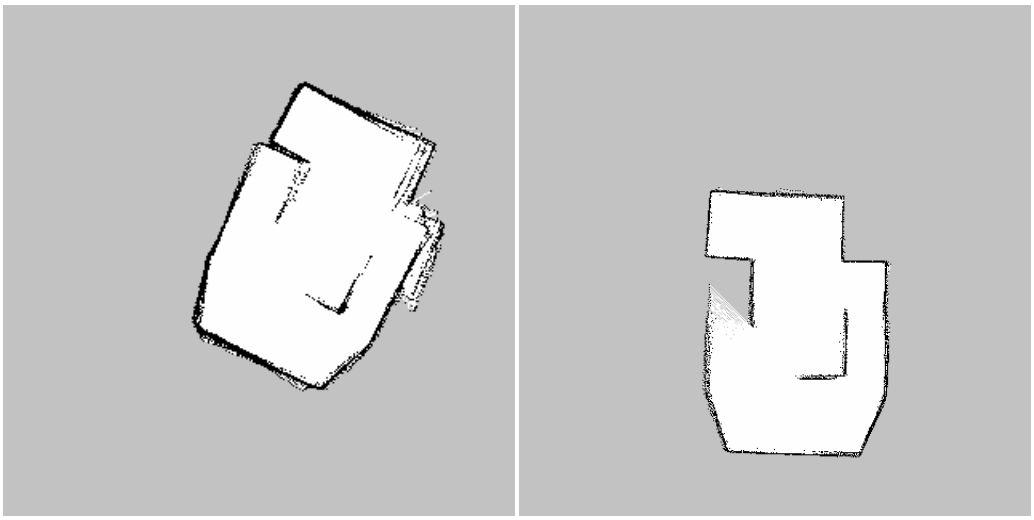


Figure 2.3: Exploration restart

A frontier goal is given every 10 seconds in order to minimize failed explorations as well to reduce the rotations in movement. Rotations lead to misalignment in the laser scans, which are induced due to the recovery behaviors of `move_base`[7] when a planned navigation trajectory cannot be followed. The recovery behaviors are maintained for navigation, and otherwise the exploration would fail straightaway when a given goal cannot be reached, leading to an incomplete map. Therefore, the frontier goals are updated at the configured frequency to prevent these two cases.

Figure 2.4 shows the resultant map from the autonomous exploration depicted above, along with a map created using teleoperation. Note that the shown exploration result was arbitrarily selected from several tests, while the result shown from teleoperation represents the map selected for use in the project, which was obtained after many trials. Therefore, a comparison between these results only serves to show the viability of autonomous SLAM for the given environment.

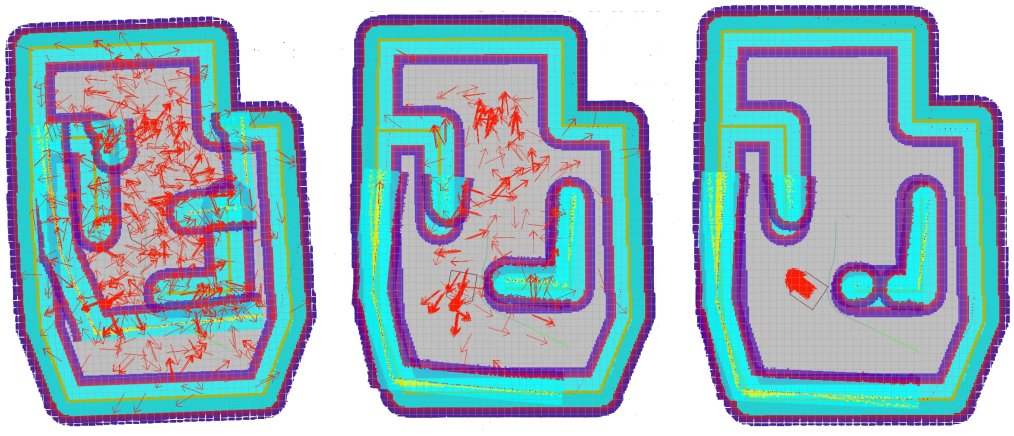


(a) Map using autonomous exploration (b) Map using teleoperation

Figure 2.4: Comparison of maps from exploration and teleoperation

2.2 Global Localization

In CatKing[8], we addressed the problem of global localization using the particle filter implementation of `amcl`. Figure 2.5 shows the working of this solution: particles are uniformly sampled in the map through an existing service call, and localization is gradually achieved as the robot moves and accumulates laser scan and odometry data. Due to the initial spread of the particles in the map, signifying complete ignorance of the robot state, patrol goals may not be followed, preventing localization and convergence. This loop was solved using `move_base` recovery behaviors, which initiate rotations upon path planning failures. The range of the laser sensor and fairly distinguishable features of the map resulted in convergence in localization in a short time.

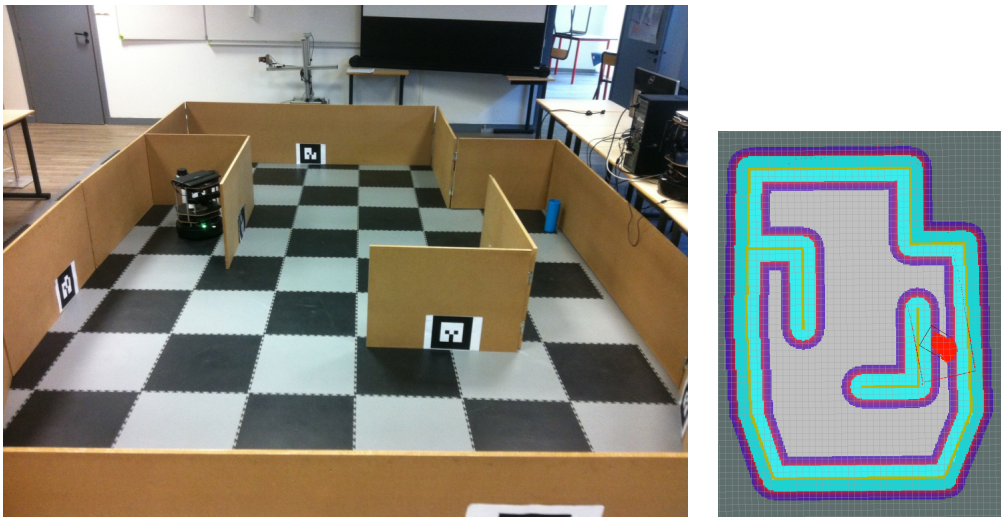


(a) Uniform belief (b) Recovery rotations (c) Convergence

Figure 2.5: Global localization: working

We also anticipated and addressed the case of erroneous localization, where due to the symmetry in the map, the particles converged to an incorrect pose. This represents the

kidnapped robot problem, wherein the robot believes in an incorrect state as in figure 2.6. The 6×6 covariance matrix from `amcl` along with a computed difference between the local and global cost maps were combined in a weighted sum to re-initiate global localization based on an empirically determined threshold. This approach worked effectively in solving a fundamental problem in robotics for the given setup.



(a) In arena (b) In map

Figure 2.6: Erroneous localization

The scope was to investigate a theoretical solution to the case of erroneous localization, which was carried out in this project. The theory of Probabilistic Robotics[9] suggests the addition of random hypotheses in the map as a function of the discrepancy between the static map and laser data. The additional poses may be derived from the measurement model, thereby placed at locations conforming to the sensor data. The particle weights indicate localization accuracy, and may be used to determine the addition of such particles. To mitigate the effects of dynamic obstacles or momentary sensor noise, a short-term as well as long-term average of the weights may be maintained, as:

$$w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow}) \tag{2.1}$$

$$w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast}) \tag{2.2}$$

where α_{slow} and α_{fast} are learning rates for the short and long term averages, respectively, and w_{avg} is the mean of current particle weights. The resampling adds random particles with a probability $max(0.0, 1.0 - \frac{w_{fast}}{w_{slow}})$. As a result, a lower short-term likelihood, corresponding to lower confidence in localization, leads to the addition of random particles.

The `amcl` node is an implementation of this algorithm, and provides the rates of the two averaging filters as parameters. The tuning of these parameters was investigated to resolve erroneous localization. A suggested configuration of $\alpha_{slow} = 0.001$ and $\alpha_{fast} = 0.1$ added few random particles, and proved of little consequence in the localization process for the given environment. A configuration of $\alpha_{slow} = 0.5$ and $\alpha_{fast} = 1.0$, the

maximum values permissible, regularly added many random particles, as shown in figure 2.7. Even so, the recovery was unreliable, and very slow. The parameters could not be configured to attain reliable and fast recovery, which our previous solution provided. The selection among the previous solution or the current `amcl` configuration to maintain in this project remains to be determined.

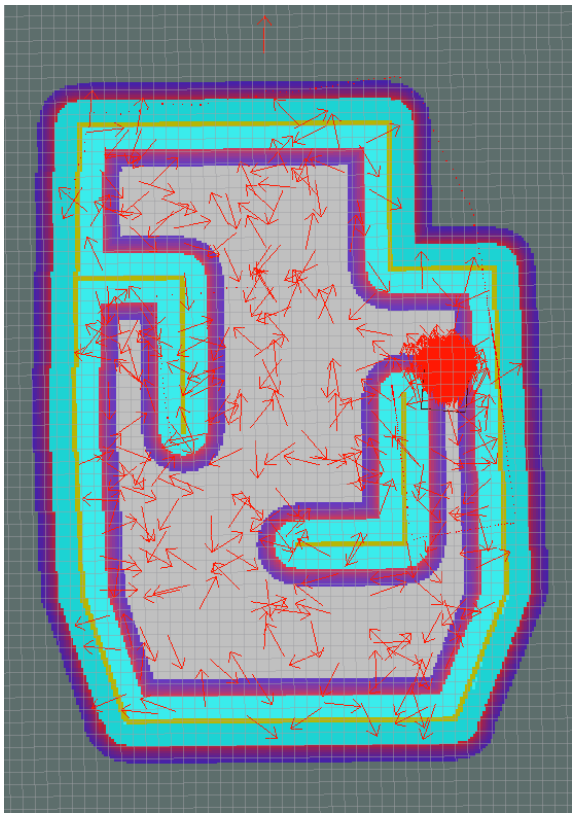


Figure 2.7: Addition of random hypotheses in map

2.3 Surveillance

Surveillance consists of patrolling the arena in a defined sequence of locations and collecting data of interest, such as images from the camera stream.

2.3.1 Patrol

Figure 2.8 shows the seven poses defined in the map for patrol. Each goal pose is maintained for a configurable period (10 seconds by default). The patrol was implemented in the driver node as a slight modification of the patrol script provided in `rbx1_nav`[10]. For effective patrol, the navigation parameters were tuned to the robot and the map. The following criteria were used in configuring the parameters:

- Safety: avoiding collisions with the static environment and minimizing chances of collision with dynamically added obstacles.

- Smooth trajectories: reducing rotations and oscillation, and following the global path planned.

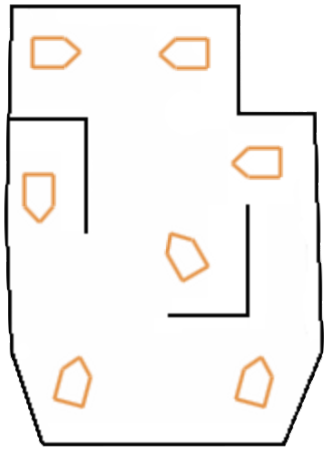


Figure 2.8: Patrol goal poses

The four navigation configuration files and their configured parameters are detailed below, followed by the reasoning behind such configuration:

1. `base_local_planner`:
    - `recovery_behavior_enabled`: true
    - `clearing_rotation_allowed`: true
    - `max_vel_x`: 0.2
    - `max_rotational_vel`: 0.5
    - `acc_lim_x`: 1.0
    - `acc_lim_th`: 1.0
    - `pdist`: 0.8
    - `gdist`: 0.6
2. `costmap_comom_params`:
    - `footprint`: modeled as shown in figure 2.9
    - `inflation_radius`: 0.3
3. `global_costmap_params`:
    - `update_frequency` : 3.0Hz
    - `publish_frequency` : 3.0Hz
    - `resolution`: 0.02
    - `observation_sources`: scan
    - `scan`: `marking`: false, `clearing`: false

4. `local_costmap_params`:
    - `update_frequency` : 3.0Hz
    - `publish_frequency` : 3.0Hz
    - `resolution`: 0.02
    - `width`: 3.0
    - `height`: 3.0
    - `observation_sources`: scan
    - `scan`: `marking`: true, `clearing`: true

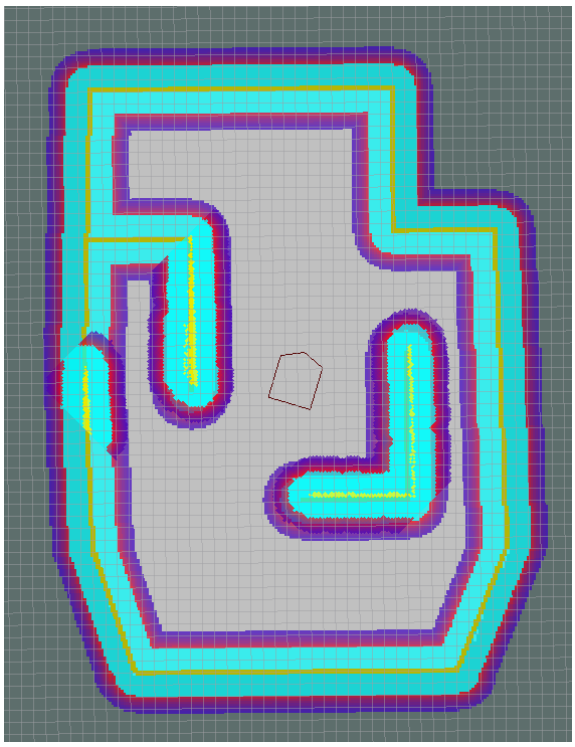


Figure 2.9: Robot Footprint

The recovery behaviors and clearing rotation provide motion required in global localization, and are also useful in navigation to handle path planning failures. The reduced motion velocity and acceleration leads to both safer and smoother navigation. A higher `pdist` parameter gives a higher importance to following the global plan. The robot footprint models the robot more accurately by accounting for the laser scanner, and also adds a safety margin through the larger polygon design. The cost maps have an update and publish frequency considering the system performance. The resolution is set to match the map.

In CatKing[8], a problem encountered during patrol was the accumulation of laser scans in the cost map, as in figure 2.10. This happened as laser scans marked obstacles often incorrectly due to the errors in rotational odometry, which were not cleared through ray-tracing. As a result, these marking remained the cost map, preventing navigation to these areas. A radical solution was implemented there by clearing the cost maps at regular intervals.

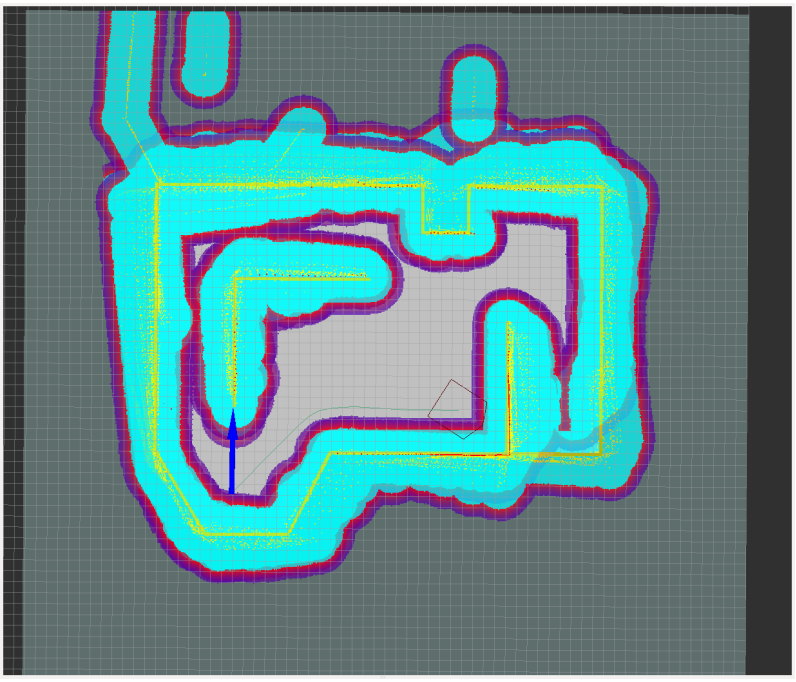


Figure 2.10: Laser scan accumulation: problem encountered in CatKing[8]

In this project, a simple solution of configuring the navigation parameters is applied. The requirement is to have a static global cost map, along with a local cost map, which may be marked and cleared through laser scan and ray-tracing, respectively. This was achieved by specifying the `marking` and `clearing` parameters independently for each cost map, as given above. As a result, the global cost map is obtained from the map server and inflated to propagate the inflation radius cost, while markings and clearings are only performed on the local cost map. A subtle documentation detail here is that the parameter `static` in the cost map[11] configuration refers to the source of the map, and could be renamed `external` to be less misleading. The local cost map was also reduced in size to ensure ray-tracing could clear markings effectively and completely.

2.3.2 Data logs

The following data are of interest for surveillance in this project:

- Camera images
- Depth images
- Camera info
- Laser scans
- Odometry
- Object identifications
- Transforms

The clear choice to record such heterogeneous data was in a ROS bag, which has the following benefits:

- Scale: simple and convenient addition and removal of topics.
- ROS abstraction: as ROS is agnostic to the source of the data, playback from a bag could be used to simulate an entire system run. Specifically in this project, offline processing is performed using data from bag files.
- Data formats: the different data types are managed natively in a bag file.

A drawback is the relatively large size of bag files. Table 2.1 shows the large size ($2.3GB$) of a test bag recording the shown topics. Using compressed image topics considerably reduces the amount of data. Still, the size is substantial given that additional topics are recorded for a longer period in test runs. However, the size of the recorded data is not considered as a major constraint in this project. In the depicted scenario, the laptop would be expected to possess sufficient resources to store the desired data, as the netbook does in the current setup. As such, no efforts were made for compressing the recorded bag files.

Topics	Messages	Duration	Size
/camera/depth_registered/image_raw	1597	53.3s	2.3GB
/camera/rgb/camera_info	1598		
/camera/rgb/image_rect_color	1598		
/tf	4497		
/camera/depth_registered/image_raw/compressed	1743	58.6s	84.7MB
/camera/rgb/camera_info	1759		
/camera/rgb/image_rect_color/compressed	1758		
/tf	4712		

Table 2.1: ROS Bags: Data Size

2.4 Manual Override

Overriding behavior for manual motion control is achieved through command velocity multiplexing. The command velocity multiplexer[12] is an instance of a mux[13] node, which subscribes to multiple command velocity publishers and outputs a single command velocity to the mobile base. This is depicted in figure 2.11, where three command velocity topics are multiplexed to result in a single command velocity published to the mobile base.

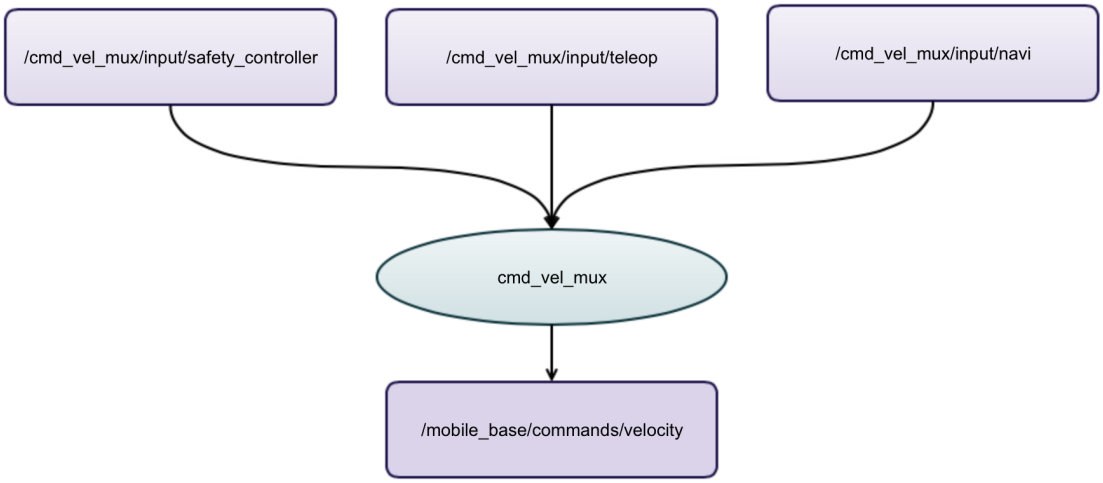


Figure 2.11: Command velocity multiplexer

A configuration file sets the priority for multiplexing, and the standard configuration was maintained, which receives input from three sources in decreasing order of priority:

- Safety controller
- Teleoperation
- Navigation

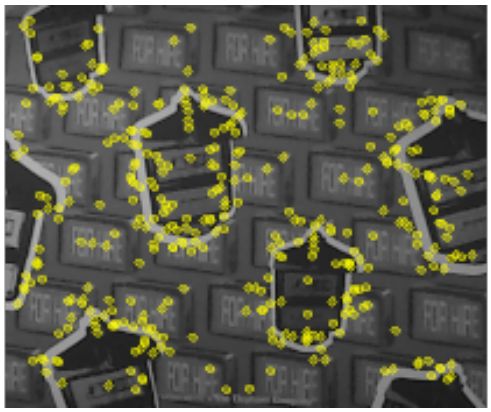
Figure 2.11 also depicts the respective command velocity topics published to by these sources. The joystick is assigned a higher priority than the navigation to result in the desired overriding behavior. The safety controller supersedes both teleoperation and navigation, and is active through the bumper and/or wheel drop sensor, depending upon its setup and inclusion.

Since the joystick and navigation nodes run simultaneously, an activation signal was implemented in the joystick node to override navigation. Otherwise, the joystick node would continuously publish a command velocity, equal to zero by default, and the navigation would always be overridden. This way, the joystick is active only when the designated button is pressed. Such functionality, with a similar implementation exists in the standard `turtlebot_teleop`[14] package, but was realized only after completing this implementation.

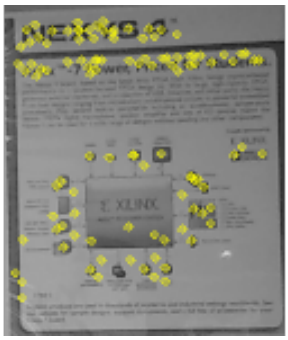
An alternative to this approach is to select the command velocity source through state transitions. While the state transition could also be achieved through a joystick input, and thereby lead to overriding behavior, the command velocity multiplexer preserves the state of the system in a simple manner. Only the command velocity published to the mobile base is interchanged upon the joystick activation, while the rest of the system remains as is. Consequently, the navigation velocities are still published, albeit overridden by the joystick, and resume control once the joystick is deactivated. This was exactly the desired behavior in this project. Also, note that multiple sources publishing to a single command velocity topic leads to jerky motion due to the differing velocities and publish rates.

2.5 Object Recognition

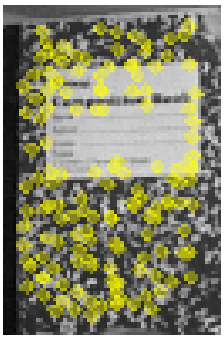
The `find_object_2d`[15] package interfaces several feature detectors and descriptors in `OpenCV` for object recognition. An object database was created, comprising four selected objects, shown in figure 2.12. The objects were selected to contain texture in order to facilitate feature detection and matching. The default `SURF`[16] algorithm was used as both the feature detector and descriptor, and figure 2.12 also shows the detected features for the selected objects.



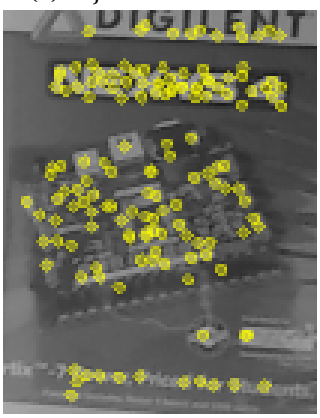
(a) Object 1



(b) Object 2



(c) Object 3



(d) Object 4

Figure 2.12: Object database

The implementation handles multi-object detection, and a result is shown in figure 2.13. All shown objects are successfully detected despite partial occlusion. Furthermore, although objects 1 and 3 appear similar visually, the matching algorithm correctly distinguishes them. The object contours are computed based on a homography estimation, which relies on planar objects. The recognition performed reliably in the experimentation setting, with no observed false positives. The performance is ultimately dependent on the feature algorithm, and the results could vary using other algorithms or in different environments. Note that the result in figure 2.13 was obtained through the user interface of the package for illustration purposes, and only the node is used otherwise.

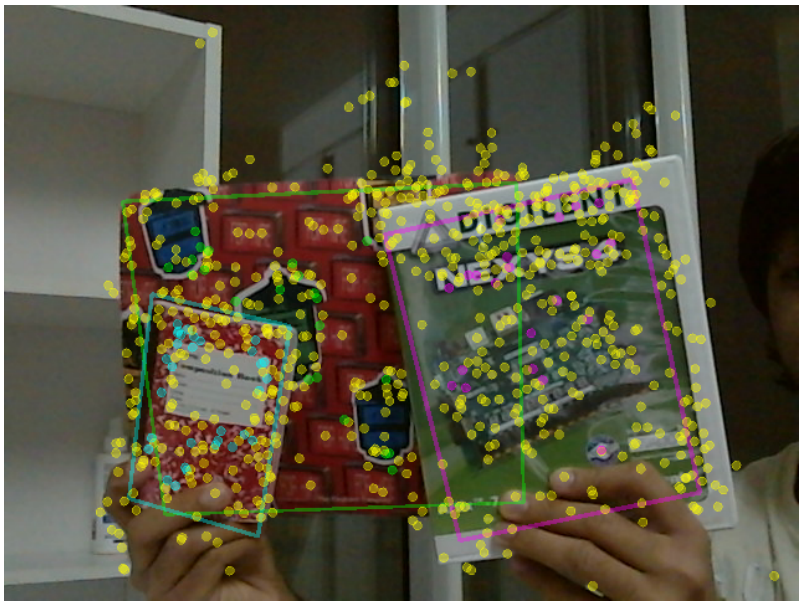


Figure 2.13: Multi-object recognition

Using registered depth images, the pose of the detected object with respect to the camera may also be determined, as shown in figure 2.14. The package also utilizes transforms through `tf`[17] for a representation in the map reference frame, which is of greater interest in the depicted scenario. Although `RVIZ` is configured to display the detected objects in the map frame, the close distance required for detection seldom enables it. All objects are under $25cm$ in both width and height, and object 3 is considerably small, a little over $10cm$ in its maximum dimension. In fact, objects 2 and 4 are opposite faces of the same FPGA board. As such, the objects are identified when fairly close to the camera, whereby their pose is difficult to obtain.

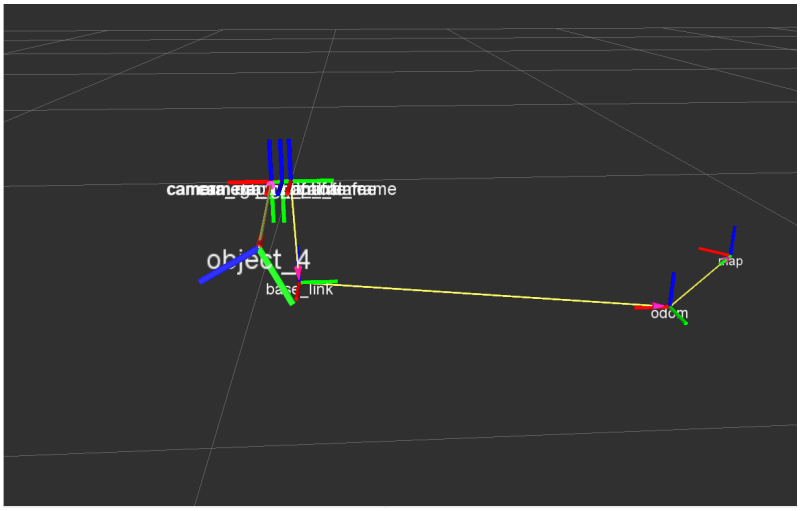


Figure 2.14: Object position in camera reference frame

2.6 Hazard Detection

Hazard detection was implemented as a particular case of object recognition. The object shown in figure 2.15, or object 4 in the database of figure 2.12, was selected to represent danger, and its detection triggers an emergency routine. This routine consists of halting patrol, and evacuating to a defined emergency exit. The preemption was implemented through inter-process message passing between the object recognition and driver nodes. Further details are given in section 2.9.

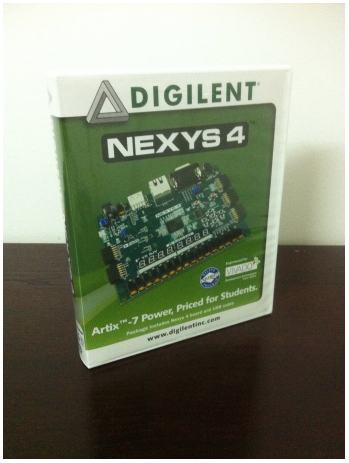


Figure 2.15: Object representing hazard

The hazard detection could be adapted to the environment, through different object models or even the data representing a hazard, such as sound or other sensor signals. Variables such as the distance to defined exit points in the map may also be considered in the emergency routine in a more realistic scenario. Here, the purpose is to demonstrate the preemption of patrol and the emergency routine in a simplistic scenario.

2.7 RGB-D SLAM

Two packages implementing RGB-D SLAM were explored:

1. `RTAB-MAP ROS`[18]
2. `RGBDSLAM v2`[19]

Both worked well in tests using a hand-held Kinect and relying on visual odometry to reconstruct my apartment. `rtabmap_ros` was selected as it provides a use case incorporating robot odometry and scan data as depicted in figure 2.16, whereas `RGBDSLAM v2` relies entirely on visual odometry. Given the largely texture less arena, using the robot odometry is crucial for registration and reconstruction. Additionally, `rtabmap_ros` is very well documented, with several tutorials for different use cases.

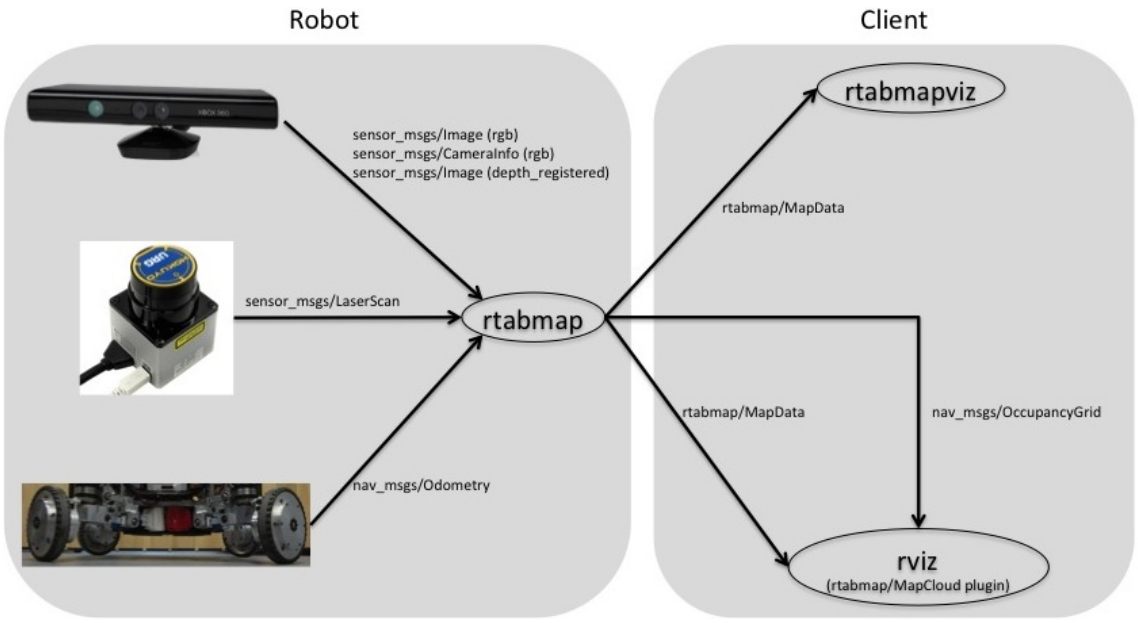


Figure 2.16: RTABMAP: robot use case

The provided user interface, `rtabmapviz` includes convenient functionality, such as saving the point cloud and the aligned laser scans. This way, the point cloud may be exported and processed in an external application, such as MeshLab[20]. An offline reconstruction of the arena from a recorded bag file using a default configuration is shown in figure 2.18. This screen shot of the reconstruction was also taken in MeshLab, where the point cloud was exported. And figure 2.17 shows the aligned laser scans forming the map of the arena.

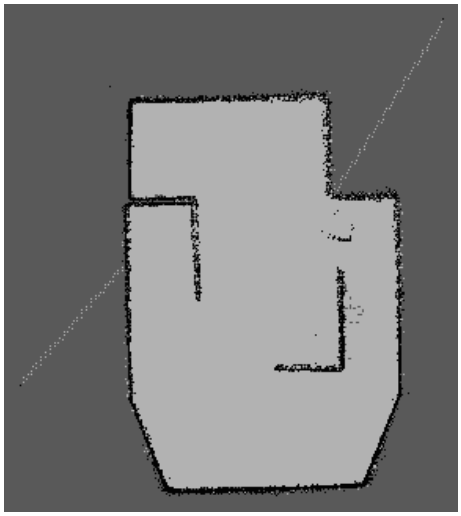


Figure 2.17: Aligned laser scans

Figure 2.18: 3*D* reconstruction of the arena

The package provides several additional functionalities, such as a localization mode, which aligns the created map and point cloud upon loop closure, as well as multi-session mapping, which could be very useful in the depicted scenario.

2.8 Speech Synthesis

The `sound play`[21] package includes capabilities to play sounds, such as speech or music. A sound client node was run on the netbook, which subscribes to the `/say_this` topic. The messages (of string format) obtained on this topic are then spoken by publishing to `/robotsound`, as is standard with the `sound_play` node.

Expected behavior:

- The messages received on the `/say_this` topic are synthesized to speech unless a `playsong` message is received, which results in playing a stored sound file instead.
- A sound being played is overridden when a new message is received. Particularly for object recognition, consecutive frames detecting the same object lead to such a "stutter" behavior.

In this implementation, any node requiring voice feedback independently publishes to the `/say_this` topic. While this approach could lead to overlap of sound messages,

voice feedback is mostly for demonstration purposes in this project, and not considered critical as the data saved in the logs reflects the true status.

2.9 Driver Node

Instead of a higher level task manager such as `smach`[22], concurrent processing is achieved by simply running all nodes simultaneously with inter-process communication to exchange data and simulate state transitions. The selected scenario and tasks allow this simple structure without conflicts among the different topics or nodes. In terms of a state machine, the driver node can be thought to comprise two states: patrol and an emergency routine, as shown in figure 2.19. These two states are briefly explained below:

- Patrol: the driver node runs through a sequence of patrol locations while a flag, `patrol_flag`, is enabled. Global localization is initiated upon start up, and patrol goals induce the motion required for localization convergence. The other tasks described in the data collection stage in section 1.1 also run simultaneously, including data logging, manual override, object recognition, hazard detection, and speech synthesis.
- Emergency exit: upon detecting a hazard, the patrol is stopped by disabling the `patrol_flag`, and an emergency routine of heading to a defined exit is followed. The hazard detection is signaled by the `object_recognition` node on the `/hazard_detection` topic, which is subscribed to by the driver node.

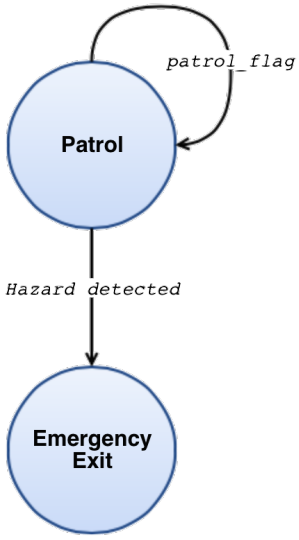


Figure 2.19: State machine

CHAPTER 3

DISCUSSION

The challenges encountered during project development are discussed, along with ideas for future work.

3.1 CHALLENGES FACED

- **RPLIDAR:** the `rplidar` process died frequently and unexpectedly. The status message indicated a segmentation fault, but efforts to determine its cause were futile. Replacing the sensor formed the solution.
- **OpenNI:** `rtabmap_ros` requires registered depth images in the computation of $3D$ point clouds. While tests using the Kinect on the workstation successfully created a $3D$ reconstruction, similar tests with the Kinect connected on the netbook failed in generating a point cloud[23]. The cause was erroneous, seemingly blank registered depth images, a problem faced by other `ROS` users as well[24]. Suggested solutions led to a failure in detecting the Kinect[25] and even removed several critical packages such as `move_base` and `frontier_ exploration`. A fix suggested in this link resulted in detection, albeit inconsistently, and with a status message yet to be understood. Thus, the image stream from the Kinect is first verified upon start up before project execution.
- **Networking:** streaming data between the netbook and the workstation was fairly slow in the lab environment, with all of us clogging the bandwidth. The poor streaming occasionally disrupted the working of nodes requiring streaming data. A few fixes were applied:
  - Subscribing to compressed image topics: vastly reduced the bandwidth requirements, as well as storage memory for recording to a bag file.
  - Saving the bag file on the netbook: since the bag is processed offline, and the netbook has sufficient memory to store the relatively large bag files, they are saved on the netbook. This is also coherent with the envisioned scenario, where unreliable or malfunctioning networking would require saving the data directly on the robot computer.
- `Roscpp` ***vs.*** `Rospy`**:** the driver node includes patrol and handles the interrupt from hazard detection. Initially, this node was implemented in `C++` as a matter of preference and familiarity. However, multi-threading was required to handle the

patrol loop simultaneously with the hazard detection subscriber. Since this is handled natively in `rospy`, the driver node was re-implemented in `Python` as a modification of the patrol script provided by `rbx1`.

- **Documentation:** the standard `ROS` tools and packages are well documented, but the independent packages vary significantly in the quality of their documentation. The documentation highly influenced the choice of packages to use, and the selected packages, such as `frontier_exploration` and `rtabmap_ros` are comprehensive in their documentation, with well organized wiki pages.

  Furthermore, fundamental tasks in Robotics, such as autonomous SLAM and global localization were developed in this project using `ROS` tools and existing packages, however, a standard tutorial or guide documenting their development is missing. There exist scattered wiki pages and answers on the forums, with some useful details, but a complete and thorough documentation could be very useful for the community.

3.2 Future Work

- **Visual servo:** Position-based visual servoing using the `demo_pioneer` package with `visp` was already configured and tested for this project. The idea was to show visual servoing as a candidate solution for object manipulation, such as plugging to a socket, through the application of auto-docking. In a passive environment a vision-based method, such as visual servoing could prove useful. Auto-docking involved navigating to way point close to the dock, and aligning to a desired robot pose with respect to a tag mounted above the dock. However, integrating navigation commands from visual servoing was difficult due to the existing patrol loop, and would be simpler with a task manager such as `smach`. Additionally, the auto-docking application was unimpressive, as the size of the tag necessitated moving very close to the dock to initialize tag detection.

- **State machine:** a task manager, such as `smach`, is required to handle autonomous functioning in an organized manner. Additional tasks, such as visual servoing would be easier to integrate in this way. The visualization capabilities of `smach` are useful as well.

- **Autonomous selection of patrol goals:** in the current scenario, the map is processed and patrol goals selected subsequent to a mapping expedition. While this is plausible in a realistic scenario as well, as this process could be performed through networking while the robot maintains its state, an autonomous algorithm to select way points in the generated map would be desirable in cases where data transfer is unreliable or unfeasible. As such, the robot would generate a map of its environment, determine goal poses to patrol, and collect data of interest without interruption or the need for networking. The $3D$ reconstruction could still be offline, unless of use during patrol.

CHAPTER 4

CONCLUSION

Explorator is influenced from the foreseeable robotics applications in exploration, search and rescue, and disaster operations. The task capabilities are modeled after such scenarios, and include autonomous SLAM, global localization, surveillance, manual override, object recognition, hazard detection, RGB-D SLAM, and speech synthesis. These and other attendant problems are active areas of research in both academia and industry, and the DARPA Robotics Challenge should serve as a catalyst in their development. The design and scalability of the current implementation could be improved by a task manager, such as `smach`. This would allow easy integration of additional tasks, such as the already tested visual servoing.

Existing `ROS` packages were mostly used and configured in this project, with a few implemented nodes where required. The ecosystem of ROS, along with the convenient tools and documentation, is compelling for robotics applications, and the driving force behind projects such as Explorator and CatKing. The Robotics course modules and projects have been highly enriching, and it was a sheer joy to spend time in the laboratory.

BIBLIOGRAPHY

[1] Turtlebot, . URL http://www.turtlebot.com.

[2] Ros hydro medusa distribution. URL http://wiki.ros.org/hydro.

[3] Darpa robotics challenge. URL http://www.theroboticschallenge.org.

[4] Mappotino. URL http://www.athoughtabroad.com/2012/05/27/mappotino-a-robot-for-exploration-mapping-and-object-recognition.

[5] Frontier exploration. URL http://wiki.ros.org/frontier_exploration.

[6] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151, Jul 1997. doi: 10.1109/CIRA.1997.613851.

[7] Move base. URL http://wiki.ros.org/move_base.

[8] D. Adlakha, S. Mohaimenianpour, and D. Nunes. Catking. URL https://github.com/Voidminded/CatKing.

[9] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, 2005.

[10] Patrick Goebel. *ROS By Example Volume 2*. Lulu, 2015. URL http://www.lulu.com/shop/r-patrick-goebel/ros-by-example-volume-2-hydro/paperback/product-22357286.html.

[11] Costmap. URL http://wiki.ros.org/costmap_2d.

[12] Command velocity multiplexer. URL http://wiki.ros.org/kobuki/Tutorials/Kobuki's%20Control%20System.

[13] Mux. URL http://wiki.ros.org/topic_tools/mux.

[14] Turtlebot teleop, . URL http://wiki.ros.org/turtlebot_teleop.

[15] Find object 2d. URL http://wiki.ros.org/find_object_2d.

[16] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In AleÅą Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision âĂŞ ECCV 2006*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-33832-1. doi: 10.1007/11744023_32. URL http://dx.doi.org/10.1007/11744023_32.

[17] tf. URL http://wiki.ros.org/tf.

[18] Rtabmap-ros. URL http://wiki.ros.org/rtabmap_ros.

[19] Rgbdslam v2. URL http://felixendres.github.io/rgbdslam_v2/.

[20] Meshlab. URL http://meshlab.sourceforge.net.

[21] Sound play. URL http://wiki.ros.org/sound_play.

[22] Smach. URL http://wiki.ros.org/smach.

[23] No point cloud genereated, . URL http://answers.ros.org/question/222637/rtabmap-ros-point-cloud-not-created-robot-mapping-use-case/.

[24] Erroneous registered depth image, . URL http://answers.ros.org/question/53706/registered-depth-image-is-black/.

[25] Kinect not detected, . URL http://answers.ros.org/question/60562/ubuntu-12042-and-openni_launch-not-detecting-kinect-after-update/.

[26] Doctoral day. URL http://le2i.github.io/doctoral-day-2015/.

APPENDIX A

PROJECT MANAGEMENT

A.1 TOOLS

Working alone on the project, only a few tools were required for project management:

- `Trello`: several lists were used to manage planning and tasks, as well as to track progress. Additionally, material and bookmarks from my personal computer were shared via `Trello`.
- `GitHub`: storing the project code and maintaining it as open-source.

A.2 DEVELOPMENT

The initial plan was to continue my work on time to contact estimation(TTC) with a catadioptric sensor[26], while fulfilling the project requirements. A new, gradient-based method for TTC estimation was developed, which was expected to perform in real-time with a high output frequency, and I was eager to implement and test it on the robot. However, upon setting up the hardware for this research, the uncertainty of progress and achieving results became apparent; the catadioptric sensor only approximated the model used in the mathematical development of the method, and produced an image with a significantly large unusable area in the center (figure A.1).

Figure A.1: Image from catadioptric sensor: large unusable area in center.

Meanwhile, I sought to address fundamental problems in Robotics and Vision, such as autonomous SLAM, global localization, and RGB-D SLAM, where the work with TTC estimation did not fit coherently. Consequently, around halfway through the project development, I decided to halt working on TTC estimation, and completely focus on the project. This detachment from the research work led to freedom in developing a fitting scenario for this project, which eventually shaped to exploration, search and rescue, and disaster operations.

Planning the project development was difficult as the time required to complete several of the selected tasks was unpredictable. More importantly, achieving success was not necessarily a given, as although I had a conception for solving problems such as autonomous SLAM, the precise packages to use and their modification to function as required were ambiguous. In this regard, the laboratory laptop and access to the Kinect were beneficial, particularly for incorporating RGB-D SLAM. Ultimately, I spent a lot of time and effort to come up with the current project, and the experience has been highly satisfying.