Improved State Estimation for a Resilient Spacecraft Executive

Riashat Islam
University College London
riashat.islam.11@ucl.ac.uk

Catharine L. R. McGhan
California Institute of Technology
cmcghan@cms.caltech.edu

Richard M. Murray
California Institute of Technology
murray@cds.caltech.edu

Abstract

Autonomous navigation systems that can operate safely without collisions in an unknown environment can have significant applications towards space exploration missions for science discoveries. In this project, we strive to further develop the initial implementation of a resilient risk-aware software architecture that can handle uncertainty in hazardous environments. We propose a model using ROS for real-time dynamic map updates using Hokuyo lidar sensors, and use generated map for autonomous navigation of the rover. We also integrate obstacle avoidance algorithms into the RSE software architecture. Our results demonstrate dynamic map updates for navigation and planning, on both simulation and practical framework using the TurtleBot robot.

1. Introduction

In recent years, there has been a growing interest to develop autonomous vehicles that can navigate the world with little human intervention. An ideal autonomous system would be able to operate safely in an unknown environment, such as in space exploration missions under uncertain environments, without any collisions with surrounding obstacles. In this project, we work towards further developing the initial implementation of a resilient risk-aware software architecture for an autonomous robot, such that it can handle uncertainty in a spacecraft behaviour within hazardous and uncertain environments (McGhan et al., 2015). The overarching goal of this project is to develop an architecture for autonomous vehicles that allows for the incorporation of intelligent systems and cutting edge planning algorithms that enable the exploration of challenging new locations without increasing risk or system complexity.

2. Background

The Resilient Space Systems Group, a collaboration between Professor Richard Murray at Caltech and others at MIT, NSA Jet Propulsion Lab (JPL) and the Woods Hole Oceanographic Institute (WHOI), is currently developing a new software architecture for resilient spacecraft operations, for robots operating in an unknown and uncertain environment (McGhan et al., 2015). In space exploration, where destinations are remote and hazardous, the operations of the spacecraft are often limited due to hazardous conditions.

This novel software architecture is called the Resilient Spacecraft Executive (RSE); it is meant to run onboard the spacecraft, allowing it to operate in real-time while taking risks into account. The software architecture is designed such that the spacecraft can adapt to its own component failures while having capabilities to navigate in an unknown environment with obstacles and also make risk-aware decisions without waiting for operator intervention. Success of this SURF project will result in the addition of new behavioral and operational capabilities to the RSE framework for ground-based autonomous vehicles supporting its ability to perform navigation safely while also making broader risk-aware decisions.

2.1. Related Work

Several works have demonstrated truly resilient behaviour to-date that demonstrated sophisticated re-
Improved State Estimation and Control

silience enabling autonomy, called the Remote Agent Experiment which was flown on the Deep Space One Mission (Nayak et al.) (Muscettola et al., 1998). Other examples of deployed resilience include autonomous navigation capability used by the Deep Impact Mission’s impactor spacecraft to assure an accurate impact with a cometary body (JPL, 2014), and the Cassini spacecrafts onboard delta-energy calculations to ensure robust Saturn Orbit Insertion, even in the presence of system reboots and failures during this critical sequence (Huy). Such missions deployed capabilities that target resilient executions of very specific critical functions. The RSE software architecture was developed with the challenge to provide resilient autonomous behaviors across the entire system and its mission.

3. Objectives and Approach

For the purposes of this SURF project, our aim was to integrate real-time mapping capabilities, map localization and path planning or obstacle avoidance capabilities into the RSE software architecture, to be used by the TurtleBot robot for a Mars Rover simulation.

3.1. Resilient Software Executive Architecture

(McGhan et al., 2015) proposes a risk-aware paradigm in the RSE architecture that is analogous to a human behaviour, as a combination of reflexive, habitual and deliberative behaviours. The reflexive layer of the RSE architecture is based on low-level control and device-embedded software for an autonomous spacecraft. The deliberative layer performs risk-aware plan execution, where the plan executive takes as input a high-level plan or goal, makes risk-aware decisions and outputs subgoals and constraints that can be further used by the habitual layer. The habitual layer is further responsible for the achievement of the control goals that are dispatched by the deliberative layer within the given constraints (such as power, time, etc), by executing actions that are determined by pre-compiled robust control policies. In our work, we focus towards the habitual layer that implements the planning algorithms in the RSE architecture.

3.2. Robot Operating System (ROS)

The Robot Operating System (ROS) is Linux based software framework that provides operating system-like services to operate robots (Morgan et al., 2009). It uses the concept of packages, nodes, topics, messages and services. A node takes data from the robot’s sensors to other nodes. Messages are the information that moves from node to node. A node sends message on a topic which is called a publisher and the receiving node has to subscribe to the topic to receive that message, hence called a subscriber. For the purposes of this project, we have developed a ROS package that is necessary to build a complete ROS based autonomous robot control system. In particular, our work involved using the ROS gmapping package for performing laser-based Simultaneous Localization and Mapping techniques (SLAM). Using ROS, we have leveraged use of the active open source sharing community and integrated packages that are commonly required for robotics applications.

3.3. ROS setup for TurtleBot

The presented work is a ROS-based control system for a TurtleBot robot for mapping and localization, using both real and self-created environments. The TurtleBot is one of the most popular research robots. To make the TurtleBot be able to perform real-time mapping and localization, a Hokuyo lidar sensor is used for scanning the environment. A Linux (Ubuntu) based computer system equipped with ROS is used to control the robot. Control commands developed that were integrated in parallel into the software architecture during the SURF project were used to teleoperate the robot during the mapping process. At the software side, we used ROS packages like rosaria, sicktoolbox wrapper, gmapping, marker saver and amcl. For simulation purposes, we interfaced ROS with both Gazebo and Stage simulation environment, and used RViz as a graphical software to observe the generated maps.
3.4. Real-Time Mapping

Continuous generation of 3D range data in real-time is essential for autonomous navigation in dynamic or unexpected existing obstacles, in order to sense and avoid obstacles. Our work involved mounting and connecting the TurtleBot robot to a Hokuyo lidar sensor that can be used to generate 3D point cloud on a moving robot, as a step towards integrating real-time mapping capability into the software architecture. We demonstrated the ability of Hokuyo sensors to perform real-time mapping on both the simulation environment in Gazebo interfaced with ROS, and the practical environment. The lidar sensor is used to generate a wide field-of-view point cloud which can then be used to generate an occupancy grid, which can be further used by obstacle avoidance and path planning algorithms. In our work, we showcase the ability of the TurtleBot rover to autonomously map out a large area in real-time.

Real-time mapping was used to create a spatial model of the environment surrounding the robot using its sensors. The pre-generated map is then used for localization and navigation. Figure 3 shows a real-time dynamic map of the environment, and we aim to achieve similar results through our work. For our work, we implemented our own scripts to be able to use it in ROS, and created ROS packages that can be efficiently integrated into the RSE architecture.

3.5. Simultaneous Localization and Mapping (SLAM)

There are three components of a SLAM problem:

- **Localization**: The robot needs to estimate its location with respect to its objects in the environment map provided.
- **Mapping**: The robot needs to map the positions of the objects that it encounters in its environment, given that the robot position is known.
- **SLAM**: Robot simultaneously maps objects that it encounters and determines its position (as well as the position of the objects) using noisy sensors.

In SLAM, the robot initial position is initially defined as the root of the world with co-ordinate space. As the robot moves, the prediction step is towards the motion model providing new estimates of its new position and also the uncertainty in the location. As new observations come in from the dynamic map updates in real-time, the robot updates its new state estimate with the new observation (Kummerle et al., 2009).

In our work, we use Simultaneous Localization and Mapping (SLAM) techniques to integrate the 3D point clouds generated by the Hokuyo sensor to create obstacle maps (Durrant-Whyte & Bailey, 2006). SLAM is the process by which the mobile robot can both generate a map of the environmental features and use it to compute its own location, all in real time. It can also be used to estimate the robot pose and orientation to improve the map position estimates of obstacles over time. Our work involves the integration of existing SLAM techniques into the RSE architecture, so that we can accurately estimate a rover’s position and produce accurate maps. For our work, we used existing...
SLAM based packages in ROS to be integrated into the software architecture.

**Occupancy Grid Map** The real-time generated maps are considered as an Occupancy Grid Map that represents the environment by a grid, and estimates the probability that a location is occupied by an obstacle.

**SLAM Gmapping in ROS** The gmapping package available in ROS (ROS, 2014) contains a ROS wrapper for OpenSLAM’s gmapping (Grisetti et al., 2007). The gmapping package provides a laser-based SLAM as a ROS node called slam_gmapping from laser and pose data collected by a mobile robot.

### 3.6. Path Planning and Bug Algorithms

For this task, we will implement locally optimal path planning and obstacle avoidance algorithms, and integrate them into the RSE architecture for use on the TurtleBot. Nominally, we used the maps generated by data from Hokuyo lidar scanner that will be mounted on the moving TurtleBot robot, as input to real-time path planning algorithms that reflect real rover operations in a simulated planetary environment. We will be using path planning and bug algorithm toolboxes from ROS and integrating their capabilities for rover obstacle avoidance into RSE. The bug algorithms were used for suboptimal planning in unknown environments; optimal trajectory planning algorithms were integrated into the architecture for cases where we may have complete, or partially-complete, global maps for planning. Algorithm 1 shows the Bug2 algorithm pseudocode.

#### Algorithm 1 Bug2 Algorithm

0: \( L_0 \leftarrow \text{init}; i \leftarrow 1 \)
0: \( \text{repeat} \) Move on a straight line from \( L_{i-1} \) to goal \( H_i \)
0: \( \text{until} \) goal is reached or obstacle is encountered at \( H_i \)
0: \( \text{if} \) goal is reached then
0: \( \text{exist with success} \)
0: \( \text{end if} \)
0: \( \text{repeat} \) Follow boundary \( Q \)
0: \( \text{until} \) \( (a) \) Goal is reached or \( (b) \) m-line is re-encountered at \( Q \) such that \( Q \neq H_i, d(Q, goal) < d(H_i, goal) \), and line \( (Q, \text{goal}) \) does not cross the current obstacle at \( Q \) or \( (c) \) \( H_i \) is re-encountered
0: \( \text{if} \) Goal is reached then
0: \( \text{Exit with success} \) \( H_i \) is re-encountered
0: \( \text{exit with failure} \)
0: \( \text{else} \)
0: \( \text{L}_i \leftarrow Q; i \leftarrow i + 1 \)

### 3.7. Dynamic Map Updates for Planning

For the purposes of our work, we worked towards developing bug algorithms that would use real time dynamically updated map data. The gmapping package in ROS publishes messages of type navmsgs/OccupancyGrid. We worked towards converting the published messages in suitable Python list of lists format, for it to be used by the Bug2 algorithm. Our implementation of the Bug2 algorithm treats the map as a list of strings, for determining free space and obstacles in the environment, and uses it based on algorithm 1 to navigate the environment. Such capabilities of using real-time dynamic map updates can also be further used by the path planner algorithms available in software architecture.

### 3.8. Adaptive Monte Carlo Localization (AMCL)

The TurtleBot uses data from moving sensors to estimate change in position over time. This is known as odometry. Robots use odometry to estimate their position relative to the starting location. The TurtleBot then tries to localize itself with respect to a known map. Therefore, TurtleBots localizes itself using odometry and laser scan data. AMCL works by figuring out where the robot needs to be on the map, in order for the laser scans to make sense. AMCL is a probabilistic localization system for a robot moving in 2D (Fox et al., 1999). It uses a particle filter to track the pose of a robot against a known map.

### 4. Experimental Results

#### 4.1. TurtleBot in Gazebo

Our initial work involved setting up the framework for running the TurtleBot robot simulation in Gazebo and Stage environment. We set up the TurtleBot simulator environment using ROS for Gazebo simulation and created an example map of the environment to navigate around with TurtleBot teleoperation operations. Work then involved exploring the Gazebo world and observing the maps on RViz environment. Majority of initial work involved becoming familiar with existing ROS packages, and setting up and running a ROS-native simulation environment (Gazebo) using pre-existing models and interfaces for the TurtleBot platform. The work documentation and steps to run
the experiments are available in (Islam, 2015)

4.2. Laser Scans Using Hokuyo Sensor

Our work initially involved setting up and generating laser scan data using the Hokuyo sensor, that is connected to the computer available in the lab. We interfaced such that ROS could connect to Hokuyo scanner, and to observe the generated scans in RViz. Figure 4 below shows the generated scan data. However, simply from using the laser scans, it is difficult to determine the entire map of the environment.

![Laser Scan from Hokuyo Laser Sensor](image)

Figure 4. Laser Scan from Hokuyo Laser Sensor

4.3. Static Map Updates

Work then involved setting up the infrastructure and integrating the mapping capabilities into the existing RSE architecture. For the purposes of our work, we used both simulation and practical environments to generate maps of the environment. However, one initial issue was that the maps generated were static and not dynamic, for it to be able to be used by the planning algorithms. For generating maps in RViz with the attached Hokuyo sensor, we created our own package that contains the required launch files. Figure 5 shows the map generated.

![Generated Map on RViz using Hokuyo Laser (Hardware)](image)

Figure 5. Generated Map on RViz using Hokuyo Laser (Hardware)

4.4. Dynamic Map Updates

Our work then involved generating dynamic map updates of the environment, such that as the robot moves in the simulated or practical environment, we could have fully observed environment maps. The results below shows the observed maps, on both the practical and simulated environments. All the results in this section are available in https://github.com/Riashat/Caltech-SURF

![Dynamic Map Updates](image)

4.4.1. Real-Time Mapping in Simulation and Hardware

![Dynamic map generated using Hokuyo sensor on Simulation Environment](image)

Figure 6. Dynamic map generated using Hokuyo sensor on Simulation Environment
Figure 7. Dynamic map generated using Hokuyo sensor

Figure 7 shows the obtained results on moving the actual TurtleBot hardware. For determining whether the generated maps are reliable, we wrote our own Python script for moving the TurtleBot in squares and in circles, keeping obstacles around it. Figure 8 shows the results obtained as the TurtleBot moved in a square.

Figure 8. Dynamic map generated using TurtleBot movement in squares

4.5. Autonomous Navigation

Once we obtained dynamic map updates, we used ROS map_saver package along with AMCL to save the map and use it for autonomous navigation of the TurtleBot. Autonomous navigation was achieved by specifying the TurtleBot pose estimates in RViz, and then use 2D goal navigation to specify the goal state in the map, for the TurtleBot to move autonomously towards the goal state in both simulation and hardware. Figure 9 and 10 shows the planning trajectory for autonomous navigation. As the 2D goal navigation was determined in RViz, the TurtleBot moves autonomously towards the goal in practice.

Figure 9. Planning Trajectory in RViz

4.6. Planning and Obstacle Avoidance

For integrating planning and obstacle avoidance capabilities into the RSE architecture, our work initially involved understanding and using the available planners in OMPL. The RSE software architecture already includes RRT* algorithm implemented into the Habitual layer using OMPL. We used the OMPL app that is available open-source to generate the simulation behaviour of planning algorithms. Figure 11 below shows the environment generated in OMPL, and figure 12 shows the RRT algorithm path achieved under the simulation environment.

For the purposes of our work, we have implemented Bug2 algorithm onto the RSE software architecture using the algorithm as shown in 1. Additionally, inspired from ??, we have created our own script that would perform simple obstacle avoidance on the TurtleBot,
and move round the obstacle that it sees, generated from the map. Figures ?? and 13 shows how the map environment format for use by the Bug2 algorithm.

Figure 10. Autonomous Navigation towards goal state

Figure 13. Integer Map of Environment to be used by HL Layer

### 4.7. Obstacle Avoidance Using Mapped Environment

Using the generated map and autonomous navigation capabilities, the next step of the project was to be able to use the maps for performing obstacle avoidance. Our work involved initial efforts towards simultaneously generating maps and navigating with it, while avoiding obstacles. Currently, the real-time mapping package, and the obstacle avoidance algorithm is implemented independently onto the RSE framework.

### 5. Discussion

There were few challenges faced towards integrating real-time mapping capabilities into the RSE software architecture. Our work involved creating, modifying and writing ROS packages to use the TurtleBot rover model with Hokuyo sensor in the gazebo simulation environment. Even though real-time mapping simulation could be done with the TurtleBot robot, there were difficulties in performing similar experiments on the other popular Pioneer 3-DX rover. We also used the Hokuyo laser sensor instead of the SICK laser sensor. With the Hokuyo sensor, we needed to make sure that the data collected from the mounting of the sensor is accurate. Accuracy of the collected data also depends on knowledge of the orientation and position of the sensor mounted on top of the rover; there were some initial calibration involved.

### 6. Future Work

Later work in this project will involve using real-time dynamic map updates for performing path planning and obstacle avoidance. As explained above, in or-
Improved State Estimation and Control

Figure 12. OMPL RRT* Simulation

In order to operate in unknown environments, autonomous systems require a map-building capability: a cyclic process of moving to a new position while sensing the environment and integrating the new data to determine the next position to move to.

The bug algorithms implemented in this work will be used for suboptimal planning in unknown environments; optimal trajectory planning algorithms will be integrated into the architecture for cases where we may have complete, or partially-complete, global maps for planning. We will also extend these algorithms by including metrics such as risk (distance from obstacles), energy, and time of traversal, such that we can take the energy model of the rover into account, and evaluate their usefulness relative to calculation time and overall performance.

Additionally, we will work towards making the TurtleBot adaptive to sensor failures. This can be achieved by using an ultrasonic sensor that can reside along with the Hokuyo sensors as fallback modes of operation. The ultrasonic sensors mounted on the TurtleBot can collect low-resolution range data than the SICK lidar; it can also only sense obstacles at very close distances. However, these sensors can still be used to create an obstacle map in 2D. Integration of ultrasonic sensors can be a part of testing the resilience of the RSE algorithms; these sensors can be used in place of the SICK sensor to test how well the RSE implementation is able to adapt to component degradation and failure. This allows for explicit evaluation of the effectiveness of the system in avoiding failure modes during operation.

7. Conclusion

This project resulted in the addition of new capabilities into the RSE software architecture implementation, used on the TurtleBot rover. Using Hokuyo sensors, we have developed real-time mapping capabilities for the TurtleBot, and have integrated the required ROS mapping packages into the RSE software architecture, which support real-time trajectory planning and obstacle avoidance algorithms. Such integration into the system provides advantages such as efficiency, flexibility and quick coverage for rovers in a planetary surface scenario. Performing 3D SLAM using continuously generated point cloud from Hokuyo sensors will give us real-time state estimation of the rovers position and orientation; further helping towards autonomous navigation. This, coupled with the planning algorithms, allows the vehicle to generate and navigate paths in real-time in an unknown planetary environment. Our work during the SURF project has therefore led to new capabilities of the RSE software architecture.

Acknowledgments

We would like to thank the Summer Undergraduate Research Fellowship (SURF) program at the California Institute of Technology and the Keck Institute for Space Studies for their generous support and funding for this project.

I would personally like to thank Professor Richard Murray and Dr. Catharine McGhan for their useful discussions, help and advice throughout the course of this project. I would also like to thank Dr. Michel Ingham and Dr. Tara Estlin from NASA Jet Propulsion Laboratory (JPL) for their useful advice and time during the course of this SURF project.

References


Fox, Dieter, Burgard, Wolfram, Dellaert, Frank, and


Muscettola, Nicola, Nayak, P. Pandurang, Pell, Barney, and Williams, Brian C. Remote agent: To boldly go where no ai system has gone before, 1998.
