Proceedings of the ASME 2017 36th International Conference on Ocean, Offshore and Arctic Engineering
OMAE2017
June 25-30, 2017, Trondheim, Norway

OMAE2017-61880

MARINE AUTONOMOUS EXPLORATION USING A LIDAR AND SLAM

Einar S. Ueland *
Centre for Autonomous
Marine Operations and Systems (NTNU AMOS)
Norwegian University of Science
and Technology (NTNU)
Department of Marine Technology
NO-7491 Trondheim, Norway
Email: einar.s.ueland@ntnu.no

Roger Skjetne
Andreas R. Dahl
Centre for Autonomous Marine
Operations and Systems (AMOS)
Norwegian University of Science
and Technology (NTNU)
Department of Marine Technology
NO-7491 Trondheim, Norway

ABSTRACT

This paper presents the implementation of a 2D-lidar to a model-scale surface vessel, and the design of a control system that makes the vessel able to perform autonomous exploration of a small-scale marine environment by the use of the lidar and SLAM. This includes a presentation and discussion of experimental results. The completion of this system has involved the development of a suitable control system that merges exploration strategies, path planners, a motion controller, and a strategy for generating controller setpoints. The system was implemented on the Robot Operating System platform, which made it possible to utilize open-source algorithms for state of the art SLAM.

INTRODUCTION

Autonomous exploration by the use of lidars on land-based robots has been successfully performed in a number of scenarios, see for example [1]. Lidars have been applied on marine vessels, such as in [2], which uses a 3D lidar, yet, it is hard to find examples where lidars have been implemented on marine surface vessels for the purpose of exploring its environment autonomously. In fact, marine autonomous exploration using a 2D lidar and simultaneous localization and mapping (SLAM) has to the authors best knowledge not been performed before.

An analog to the marine surface vessel studied in this report is ground vehicles where 2D lidars have been used extensively. In particular, several sources describe mapping by the use of lidars installed on wheeled robots by the use of the Robot Operating System (ROS) [3] and open source SLAM software, see for example [4]. These cases served as an inspiration for the main authors master project [5], from which this paper is presenting the main results. To ease the implementation of open source mapping algorithms, ROS was chosen as the programming platform of the system reviewed in this paper. The ROS platform further supports a simple integration with MATLAB/Simulink [6] where most of the control system has been designed.

The development of a system capable of exploring its environment can be viewed as a part of a recent effort by the marine community towards the development of autonomous, unmanned marine vessels. An example showing this effort is the European research project MUNIN [7].

The system and the experiments reviewed in this paper are limited to a controlled environment where there are no waves or current present. In addition walls and hinders used in the experiments are static, and in most cases vertical. Finally, the operations are performed in small-scale, meaning that there are always some objects within the range of the lidar. For these reasons, it is expected that the resulting system needs to be further developed in order for it to function at the open sea. However, given that these limitations can be overcome, several potential applications such as safe harbor-parking for smaller vessels and various oper-

*Address all correspondence to this author.

ations that require precise navigation relative to marine structures and can be imagined.

OBJECTIVES

This paper aims at providing an answer to the following question:
Given the installation of a lidar on an unmanned surface vessel, how should a guidance, navigation, and control system be designed, making the vessel capable of efficient and autonomous exploration?

In order for the vessel to perform this exploration, it is vital it can handle the following four competences: (Italic font mark the section in the paper where the item is addressed)

1. Build a map, while simultaneously locating itself within it. (*SLAM*)
2. Evaluate which locations it should move to in order to fulfill its mapping objective. (*Exploration Strategy*)
3. Plan obstacle-free paths to desired locations. (*Path Planner*)
4. Follow the planned path based on sensed data and inputs to its actuators. (*Motion Control* and *Velocity Control Law*)

The work presented focuses on designing a setup that performs well in experiments and it makes an effort of demonstrating the design of the system as well as the experimental results.

MAPPING OF ENVIRONMENT

The area that the vessel is operating in is to be mapped in the 2D, horizontal plane, by the use of a lidar sensor and mapping algorithms.

Lidars

A lidar, which is the sensor device to be utilized for mapping in this paper, is a remote sensing device that functions by emitting a laser pulse that is reflected by the object it reaches. The returning signal is sampled by vision acquisition embedded in the lidar. The lidar measures the time that the light uses to return to it, and applies this information to produce a point cloud that can be utilized for mapping and localization.

Lidars are recognized for high accuracy, allowing for fast data acquisition and for being independent of ambient light. Figure 1 illustrates how the 2D lidar applied in this thesis function. It emits a laser pulse that is reflected by a wall and sampled by vision acquisition in the lidar. This allows the system to sense its environment in the 2D horizontal plane.

SLAM

SLAM is the problem of creating and updating a map of the unknown environment, while simultaneously determining the position of the object within it.

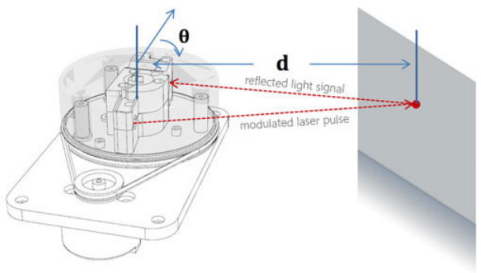


FIGURE 1: **2D LIDAR. COURTESY OF** [8]

There are several packages available in the Robot operating system that can perform SLAM by utilizing a lidar range-cloud. See [9] for an evaluation of the most popular packages. Since the Hector-SLAM package [10] is popular, well tested and does not require odometry data, this is the package that has been chosen to perform SLAM on the implemented system. The SLAM package is implemented without any changes in the internal algorithms. For a more detailed discussion on the SLAM this paper refers to [11] which discuss strategies and common algorithms for solving the problem.

Map Representation

The maps generated from the Hector-SLAM algorithms are represented as probabilistic occupancy grids, a data structure where the area which the vessel operates in is discretized into a grid. The occupancy grids are further characterized by a given grid size and resolution.

In the control system of the vessel, the occupancy grids generated from the SLAM algorithms are imported to a MATLAB script where each cell is interpreted as either an occupied cell, a free cell or an unexplored cell. Occupancy grids are a convenient form to represent detailed maps, and can be recognized in figures throughout this paper.

EXPERIMENTAL PLATFORM

The system described in this paper is implemented on the model-scale surface vessel that can be seen in Fig. 2. The vessel is round and cone-like, with diameters 548 mm and 398 mm at the top and bottom respectively. The vessel is installed with three maneuverable thrusters, the shaft of which are all placed on the circumference of a circle about the center of origin with radius 0.138 m , and they are placed with a spacing of 120 deg, i.e. symmetrically. This means that the vessel can maneuver in both surge and sway efficiently. This is an advantage for this project, as it means that the vessel's heading does not need to be considered as a parameter in the path-planning process.

As mentioned in the introduction, the control system is im-

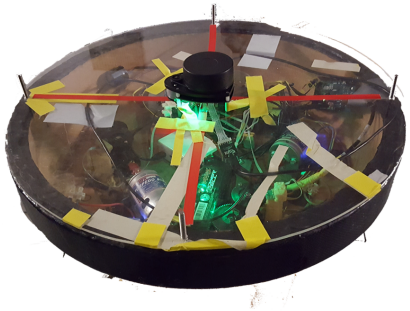


FIGURE 2: VESSEL USED IN EXPERIMENTS.

plemented on ROS. This paper will repeatedly refer to ROS nodes, which is separate processes that perform computations, and ROS topics, which are name buses which nodes use to exchange messages.

If the model vessel were to have a more conventional ship design, mainly restricted to movements in surge direction, it seems reasonable that the implementation of a separate heading controller with a steering law similar to that of [12, Section 10.3], combined with a turning maneuver for handling sharp changes in the desired heading angle would overcome the additional challenge of heading control. Thereby creating a system able to perform operations similar to those presented in this paper.

Hardware Architecture

The installed components on the system are mostly low-cost and readily available for consumers. Development of similar projects can, therefore, be expected to be performed at a relatively low cost. The hardware architecture including signal flow between components can be seen in Fig. 3.

The single-board computer Raspberry Pi 2 (RP2) running ROS on Ubuntu, is functioning as the onboard computer, capable of running multiple ROS nodes. The embedded circuit Arduino Mega, connected to RP2 via USB and is responsible for transmitting appropriate signals to the actuators. The lidar RPLIDAR is mounted on the vessel lid, is also connected to the RP2 via USB. The vessel is installed with three azimuth thrusters, each of which is driven by a separate Torpedo 800 motor. An 11.1V, three cell 640 mAh lithium polymer battery from Traxxas serves as the vessels power supply. A computer is further connected to the system via the local Wi-Fi in the basin. Finally, three Hall effect sensors are implemented, measuring the rotational speed of the thrusters, offering an opportunity for RPM feedback on the individual motors.

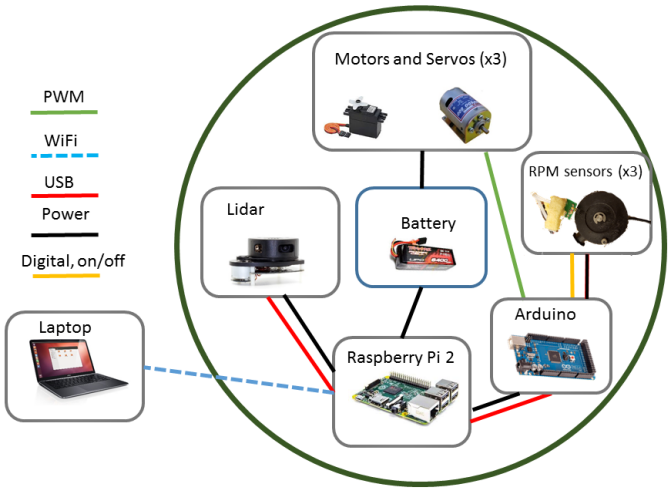


FIGURE 3: **SIGNAL AND POWER FLOW ON THE VESSEL.**

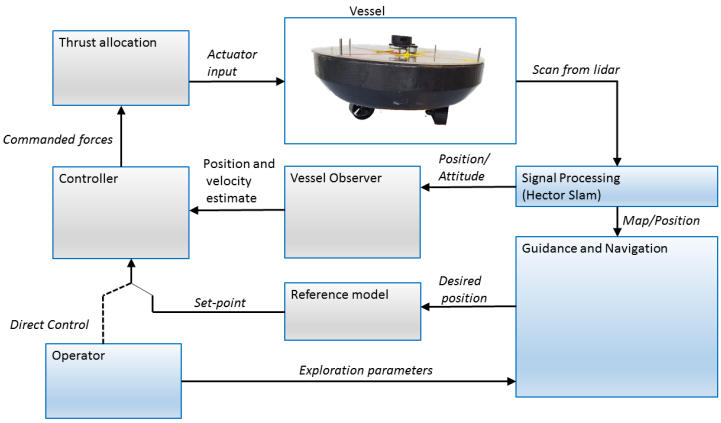


FIGURE 4: **CONTROL LOOP OF IMPLEMENTED SYSTEM.**

GUIDANCE, NAVIGATION AND CONTROL

Guidance navigation and control (GNC) refer to the system that process information from its environment and subsequent control the movement of a craft. Common strategies and algorithms for marine GNC systems are comprehensively reviewed in [12].

Figure 4 display the control loop of the implemented system and what information the individual components transmit between each other. This loop constitutes the GNC system of the vessel and is responsible for making the vessel achieve its control objective.

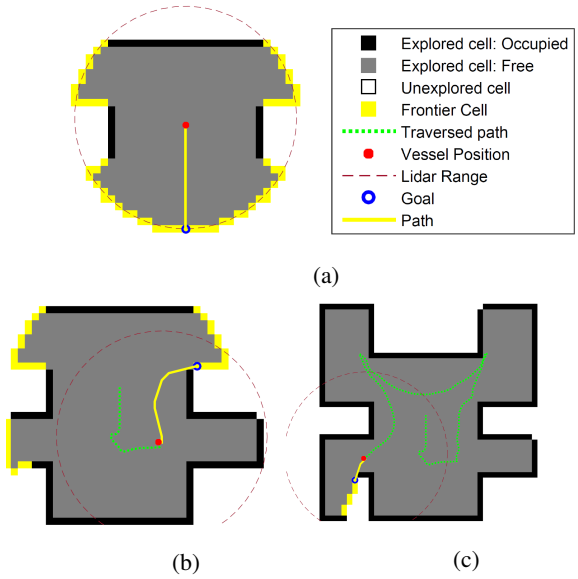


FIGURE 5: **FRONTIER BASED EXPLORATION.**

Motion Control

The goal of the implemented motion controller is to make the vessel able to track a reference setpoint. The gray boxes in Fig. 4 represent the motion control system of the vessel, and is implemented as separate ROS-node. The motion controller consists of a nonlinear observer that estimates the position and velocity of the vessel, designed according to [12, Section 11.4.1], a PD controller, a third order reference model that filters the desired position such that it is smooth and within the limits of the vessel's capabilities, and a thrust allocation scheme responsible for acting the desired generalized force vector on vessel.

Exploration Strategy

The frontier based-exploration strategy has been implemented on the system. In this approach, the robot should always move to frontiers (edges) between explored and unexplored map [13]. The method is recognized for being relatively easy to implement, for its reliability, and for being efficient in mapping new territory.

The following steps summarize how the exploration strategy is implemented to the system:

1. Identify explored cells that neighbors an unexplored cell as a frontier cell.
2. Use a path-planner to identify the frontier that can be reached with the lowest cost. The path planner generates a path to the frontier that is to be followed
3. Allow for the vessel to start following the path and repeat the process

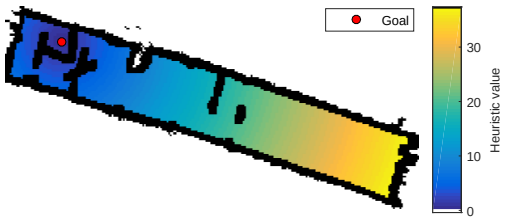


FIGURE 6: **HEURISTIC VALUE, EXAMPLE.**

Figure 5 illustrates how the simulated vessel utilizes the frontier-based exploration scheme. The simulated scenario is explored methodically in an efficient manner.

Path Planner

In order to navigate to locations determined by the exploration strategy, the A* (A star) search algorithm [14] has been implemented. The algorithm is implemented, using the center of each grid cell as a separate node that the path can visit. The algorithm associates movement between neighboring nodes with a cost and is able to find the lowest cost path between two arbitrary nodes in the grid.

In planning, the algorithm utilizes the heuristic value, which estimates the cost of the cheapest path from the investigated node to the goal. In this project, the heuristic value of each cell is set such that it equals the euclidean distance from the cell to the nearest goal node. The algorithm is always investigating the node that has the lowest combined cost and heuristic value, i.e., it prefers looking for paths where the heuristic value is low. An example the heuristic values in a scenario is illustrated in Fig. 6.

The algorithm is not reviewed in detail in this paper, however, the following should be noted about it:

1. The A* search algorithm is complete, meaning that if there exists a path to the goal node it will find it.
2. The method is admissible, meaning that it finds a best path if the heuristic does not over-predict the actual minimum cost of reaching the goal.
3. The method solves the problem by investigating the most promising nodes first, classifying it as a best-first-search algorithm.

Handling of Multiple Goal Nodes. Since the exploration method applied can yield multiple goal candidates (frontiers), the search algorithm should be able to handle multiple goal nodes. In the implemented algorithm this is realized by adjusting the heuristic such that it equals the Euclidean distance from each cell to the nearest goal node. The first goal candidate the method opens during a search is now identified as the closest goal node, from where the path is retraced back to start.

Path Orientations. In the standard A* search in occupancy grids, only the eight neighboring tiles are investigated when expanding paths from a node. This restricts the orientation of the planned path to eight directions, in increments of 45 degrees, which again leads to suboptimal paths.

A way to circumvent the restriction is to allow nodes to connect to nodes that are more than one tile away. Figure 7 illustrates which nodes are investigated when expanding the path, for connection distances between 1 and 4. Each connecting line in this figure represents a possible heading that the calculated path can have. As is evident from the figure, increasing the connecting distance quickly increase the number of possible directions. A connecting distance of one, two, three and four yields 8, 16, 32 and 48 possible orientations respectively.

While larger connecting distances increase the number of possible orientations, and in general lead to better and shorter paths, it quickly increases the complexity of calculations and thus also computation times.

Figure 8 illustrates how the calculated paths differ when using a connecting distance of one and four (the connecting distance of four is applied in the experiments reviewed in this paper). As expected, the calculated path using a connecting distance of four is both shorter and smoother than that of one.

Weighting of Cost-map. The algorithm is implemented with a scheme for weighting the crossing of cells such that the vessel prefers to keep a distance from objects. The weighting $w(s)$ of each cell is given according to the following formula:

$$w(s) = 1 + \frac{5}{0.1 + d_{\text{obj}}} \tag{1}$$

Where d_{obj} is the euclidean distance from the node to the closest occupied cell. The formula has been tuned to suit the vessel objective and dynamics.

Figure 9 illustrates a weighted grid, and the resulting path found in it. In general, paths using the weighted grid are satisfying and shown to keep a reasonable distance to objects when possible.

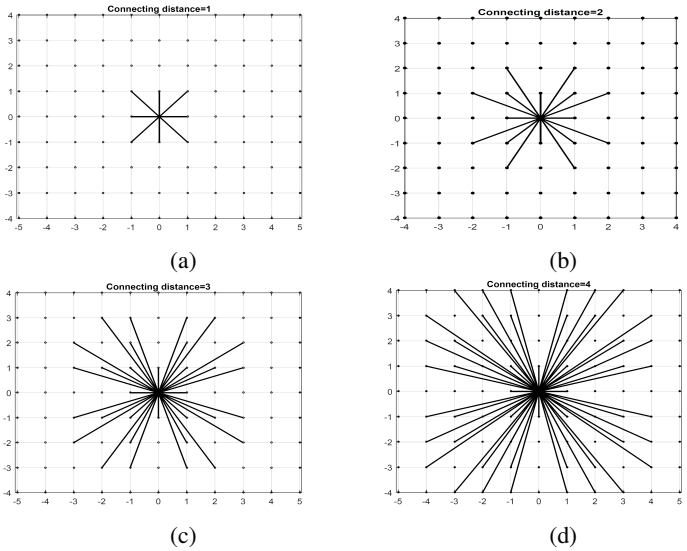


FIGURE 7: GRID CONNECTIONS USING DIFFERENT CONNECTING DISTANCES.

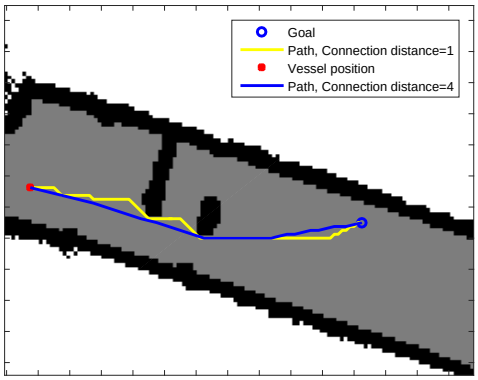


FIGURE 8: PLANNED PATH, CONNECTING DISTANCE OF ONE AND FOUR. .

Map Processing

In order not to collide, all cells within the area that the vessel spans to need to be free. In addition, cells in an additional safety distance around the vessel should be free, such that it has room to maneuver. To ensure this, all cells within a predefined distance to an occupied cell are considered inaccessible and thus labeled as occupied. This process is called inflating the map. In the experiments seen in this paper, the inflation distance is set to 0.4 m.

In addition to the inflation process, the map is processed by reducing free cells to only reachable cells, and by assuming that

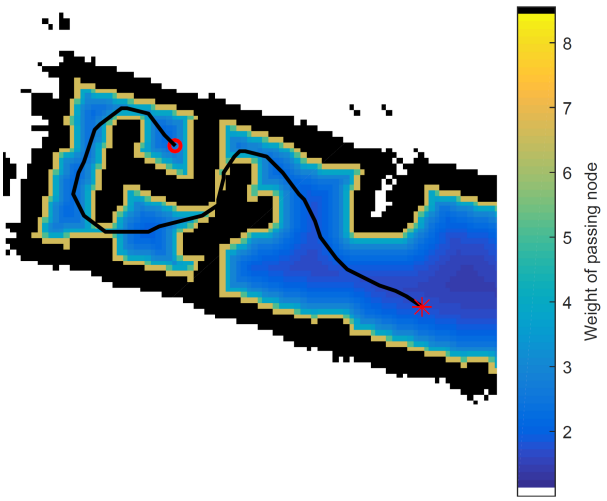


FIGURE 9: **PATH IN WEIGHTED GRID.**

cells not identified by the SLAM algorithms in the close vicinity of the vessel are free.

Figure 10 illustrates both the steps of the map processing described in this section (a-d) and the previously discussed path generation process (e-h). The figure is thus serving as an illustration of the full process from importing the map from Hector-SLAM to the generating a path for the vessel to follow.

Velocity Control Law

The velocity of the vessel is controlled by adjusting the set-point that the motion controller receives. This procedure is in this paper labeled as a velocity control law. Together with the exploration strategy and the path-planner, this constitutes the guidance and navigation block seen in Fig. 4

A list consisting of the first 128 discrete positions in the planned path is sent from the exploration node to the ROS-framework after each iteration. Based on this list, the vessel's position, and the nearest object, as recognized by the lidar, the velocity control law generates the controller setpoints. The ROS node that performs this operation is running independently on the exploration node, which allows the commanded setpoints to be updated at a high frequency, even if the path planner only updates the chosen path every few second.

The velocity control law can be described as follows:

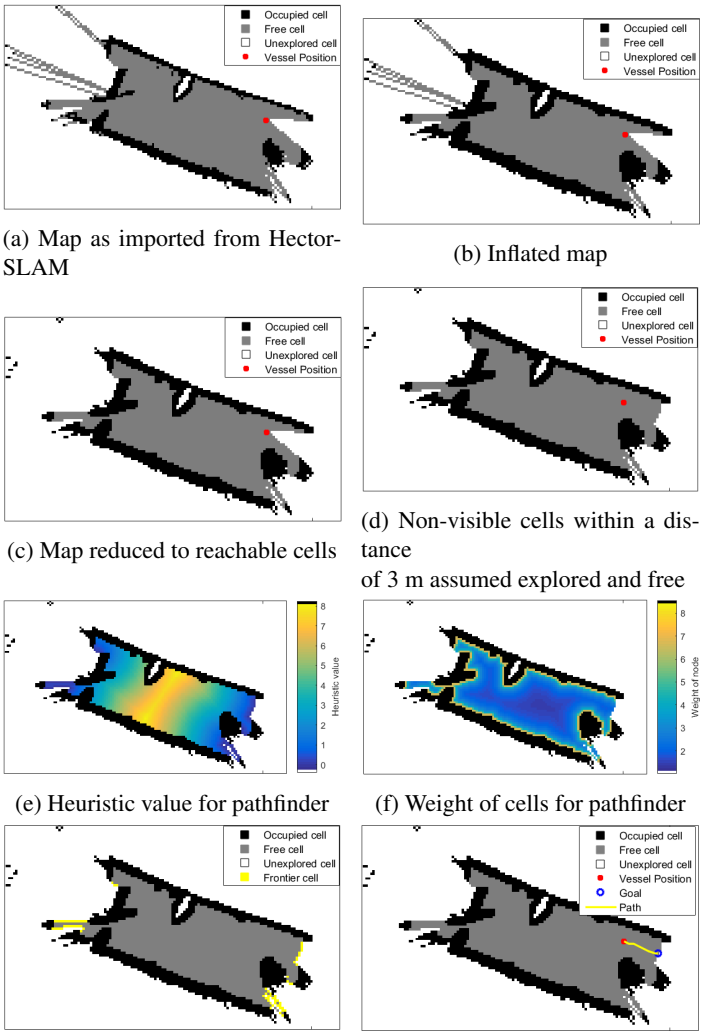
1. **Rediscretizing Path**. The imported path vector is re-discretized to an array with a step size of $r_q = 0.01$m. The discrete points are labeled $p_k, k \in (1,2,\cdots,N)$. In Fig. 11 the re-discretization is performed between (a) and (b).

2. **Identify Closest Point on Chosen Path**. The discrete point number l, that is closest to the vessel's current position is identified. This is the yellow dot in Fig. 11b

3. **Generate Setpoint Distance.** The lookahead distance, which is the distance r_s from the point p_l that one should iterate forward in the path to find the setpoint p_s is now chosen based on the distance to nearby objects:

$$r_s = \begin{cases} 0.1, & \text{if } d_{\min} \leq 0.3, \\ d_{\min} - 0.2, & \text{if } 0.3 < d_{\min} < 1, \\ 2d_{\min} - 1.2, & \text{if } 1 \leq d_{\min} < 2.1, \\ 3, & \text{if } d_{\min} \geq 3. \end{cases} \quad (2)$$

where $d_{\min}$ is the shortest distance that to an object, as



(a) Map as imported from Hector-SLAM

(b) Inflated map

(c) Map reduced to reachable cells

(d) Non-visible cells within a distance of 3 m assumed explored and free

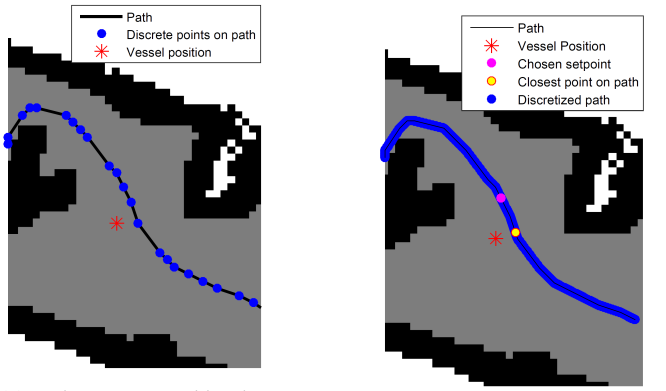
(e) Heuristic value for pathfinder

(f) Weight of cells for pathfinder

(g) Frontiers indicated

(h) Chosen path

FIGURE 10: **MAP PROCESSING AND PATH GENERATION STEPS.**



(a) Path as generated by the path planner

(b) Rediscretized path

FIGURE 11: **GENERATION OF SETPOINT.**

recognized by the lidar.

4. **Iterate to Setpoint on the Chosen Path**. The discrete point identified as the closest to the vessel's position is now used as a reference point from where to calculate the new setpoint. The setpoint is found by iterating n discrete points forward in the path from the identified closest position.

$$n = \text{round}\left(\frac{r_s}{r_q}\right). \tag{3}$$

Where round$(\cdot)$ sets $(\cdot)$ to the nearest integer value. The setpoint is finally set as

$$p = p_s, \tag{4}$$

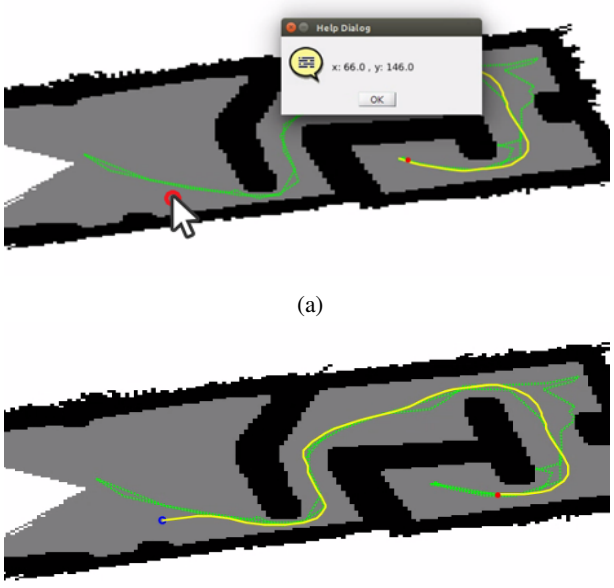
where $s = \min(n+l, N)$.
The setpoint found by this procedure can thus be sent to the motion controller. In Fig. 11b the setpoint sent to the motion controller is the purple dot.

Operator Interaction

During operations, a window that displays a figure of the map, together with position, planned path and path destination is automatically updated on the operator computer. This window is interactive, and the user can utilize it to toggle between automatic exploration and navigation to a known location on the map. This display can be seen in Fig. 12

GNC System Implementation

In the implemented ROS architecture there are several nodes, performing different tasks. An overview the network of



(a)

(b)

FIGURE 12: **USER INTEFACE (ZOOMED).**

nodes and topics in the ROS architecture can be seen in Fig. 13. This figure also displays the physical units each node is run on. Only the topics that are vital for the control of the vessel are included in the figure. Thus, topics that are responsible for parameter-setting or monitoring the system is not included.

The following nodes are present in the system: (*ROS topics are identified through italic, while ROS nodes are identified by bold style*)

1. **RP-Lidar**. Driver for the lidar. Generates a point cloud of range data (*/Scan*).
2. **Hector-Slam Nodes**. Performs SLAM that generates an occupancy grid (*/Map*) and the vessels position within it (*/Poseupdate*).
3. **Scan2SetPointDist**. Calculates the desired line of sight (*/SetPointDistance*) for the path follower.
4. **Path2Setpoint**. Generates the setpoint (*/Setpoint*) on the planned path.
5. **Hector2VesselPos**. Transformation of the desired position from Hector-SLAM (*/Poseupdate*) to vessel coordinates (*/Position*).
6. **MotionControl**. Controls vessel to desired position by setting desired control input of the actuators (*/Thrust1,/Thrust2,/Thrust3,/a1,/a2,/a3*).
7. **ROS-Serial**. Transmitting the appropriate PWM signals, as calculated by the control system to the motors and servos of the system.
8. **ExplorationPathplanning-Node**. Generate reference

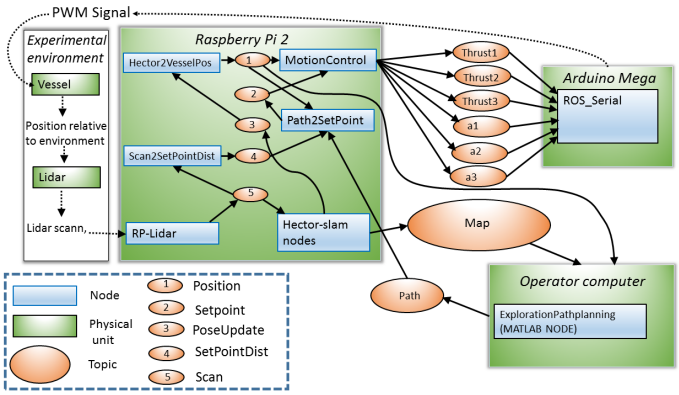


FIGURE 13: **ROS NODE AND TOPIC INTERACTIONS IN THE IMPLEMENTED SYSTEM.**

tracks (*/Path*) for the vessel to follow.

The RP-lidar node [15], and the Hector SLAM node [16] are open source packages that have been imported to the system, the Arduino code is written in C, while the other nodes have been written in Simulink/MATLAB and built by MATLAB C++ code-generation. The ExplorationPathplanning-node, was run in MATLAB on the operator computer, during experiments as opposed to being compiled in C++ and installed on the RP2.

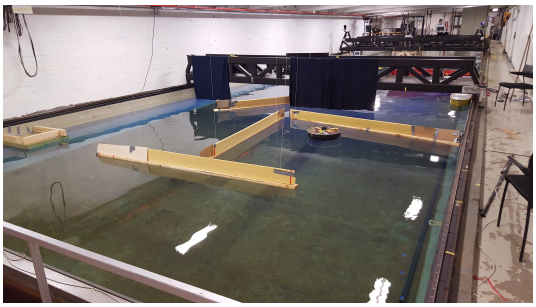
EXPERIMENTAL RESULTS

During the experiments, the vessel is operating in the Marine Cybernetics Laboratory, a basin at NTNU with length and width of 40 m and 6.45 m respectively. To demonstrate the vessels capability of autonomous exploration, varying obstacles were present in the basin during operations. These included camera mounts, other vessels, and hinders placed on wooden planks. During operations, all objects were kept static, which presently is a requirement for the SLAM algorithms to perform properly.

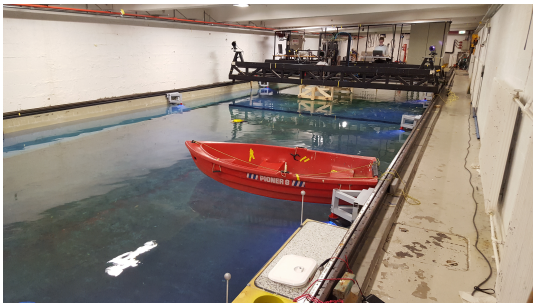
The experiment was conducted by placing the vessel in a given location in the basin, from where the software system was initialized. At initialization, the software system had no information about its surrounding environment, and its mission was to explore its environment in an efficient manner.

Two experiments are presented in this section. In Experiment 1, the vessel starts in the middle of the basin and explore the full basin, while in Experiment 2 the vessel starts in the labyrinth-like section and explores towards the center. The obstacles in the basin are set up slightly differently in the two experiments.

Both experiments were performed using an occupancy grid with a resolution of 0.2 m per cell. A video of the experiments has been made available online [17].



(a) Labyrinth section



(b) Middle section

FIGURE 14: **LABORATORY SETUP EXPERIMENT 1.**

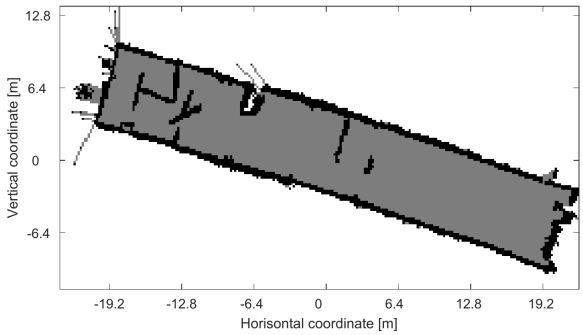


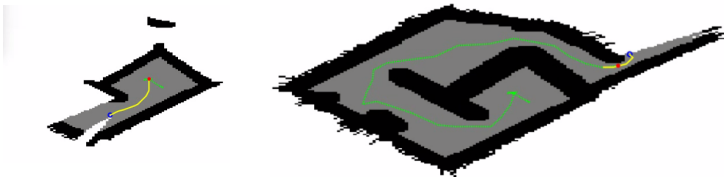
FIGURE 15: **EXPLORED BASIN, EXPERIMENT 1.**

.

Experiment 1

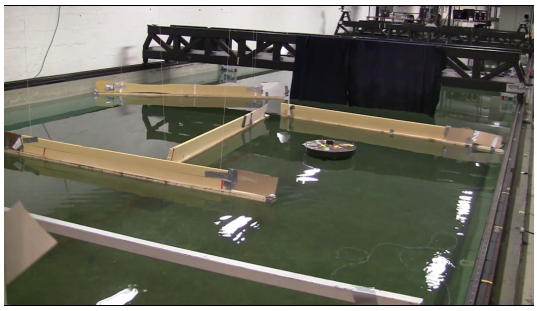
In this experiment, the whole basin was successfully explored. The vessel mostly behaved as expected, but a few incremental adjustments were still performed in the algorithms prior to the next experiment.

Images of the basin as set up in the experiment can be seen in Fig. 14, while the fully explored basin from this trial can be seen in Fig. 15.

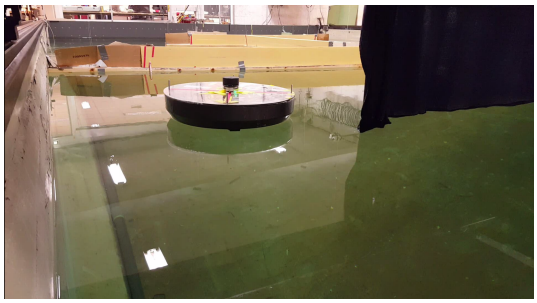


(a) Map, Position A

(b) Map, Position B



(c) Vessel, Position A



(d) Vessel, Position B

FIGURE 16: **OPERATION, EXPERIMENT 2.**

Experiment 2

This experiment yielded very satisfying results, with the vessel exploring the scenario without problems. Figure 16 illustrate the vessel with the explored map at two instances during the exploration.

The SLAM algorithms are able to recognize most objects successfully, however, it had some issues in recognizing the wall on the very left hand of the basin. This wall is a part of the basins wave generator that has a smooth, not purely vertical surface. The effect can be seen in Fig. 16b, where after halfway crossing the width of the basin, the vessel took a detour and got quite close to the wall, before it identified it as an obstacle and moved on.

Speed Analysis. By investigating the speed of the vessel, which is provided in Fig. 17 one can study how the velocity control law is performing. The vessel is adapting its setpoint, and thus also its speed according to the distance to hinders. This helped to ensure that the vessel never came close to colliding,

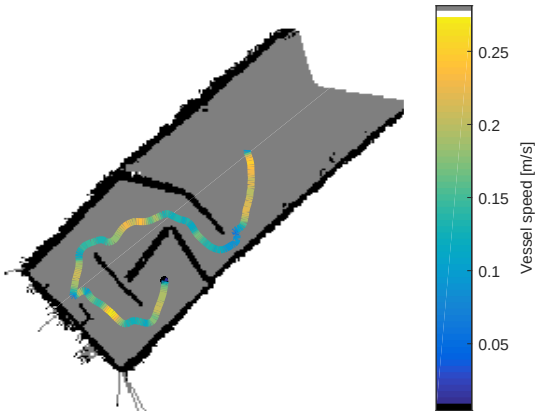


FIGURE 17: **VESSEL SPEED, AS ESTIMATED BY THE OBSERVER, EXPERIMENT 2.**

while at the same time it efficiently explored the basin. Although the speed also varies as a natural result of the vessel changing directions, it is clear from the figure that the vessel is adjusting its speed according to the distance to nearby objects. In particular, this is evident when the vessel passes the narrow passage, corresponding to position B in Fig. 16. In this section, the vessel slows down to about 0.07 m/s. Once the vessel has passed it and gets into the open area, it quickly speeds up to about 0.25 m/s.

It should be noted that as an effect of very low damping in yaw and no stabilizing keels, the vessels performance in reference tracking is limited. For this reason, the controller gains have been kept relatively low, resulting in low operating speeds ranging from 0 to 0.3 m/s. With better control of the vessel, the gains could have been set higher, without risks of crashing, and the basin could have been explored faster.

CONCLUSIONS

Summarized, the implemented system merge suitable SLAM algorithms, an exploration strategy, a path planning strategy, a motion controller, and a velocity control law to a well-functioning GNC system. This system is able to perform the objectives of exploration that this project set out to solve. A benefit of the generated system is that it constructed by the use of (relatively) low-cost components. The presented system is implemented on a circular model vessel with similar performance in surge and sway. However, it is reasonable that by introducing a separate heading controller, a similar system could be implemented on a fully actuated model vessel with a more conventional ship design.

ACKNOWLEDGMENT

This work was supported by the Research Council of Norway through the Centre of Excellence AMOS, project no. 223254, and through the project no. 254845, "Real-Time Hybrid Model Testing".

REFERENCES

[1] Nagatani, K., Okada, Y., Tokunaga, N., Kiribayashi, S., Yoshida, K., Ohno, K., Takeuchi, E., Tadokoro, S., Akiyama, H., Noda, I., et al., 2011. "Multirobot exploration for search and rescue missions: A report on map building in robocuprescue 2009". *Journal of Field Robotics,* ***28***(3), pp. 373–387.

[2] Leedekerken, J. C., Fallon, M. F., and Leonard, J. J., 2014. "Mapping complex marine environments with autonomous surface craft". In Experimental Robotics, Springer, pp. 525–539.

[3] ROS-community, 2017. ROS. [Online; accessed 3-January-2017 from http://wiki.ros.org/].

[4] Verbiest, K., Berrabah, S., and Colon, E., 2015. "Autonomous frontier based exploration for mobile robots". In International Conference on Intelligent Robotics and Applications, Springer, pp. 3–13.

[5] Ueland, E. S., 2016. "Marine autonomous exploration using a lidar". Master's thesis, NTNU.

[6] MathWorks. Robot operating system (ROS) support from Robotics System Toolbox. [Online; accessed 3-January-2017 from https://se.mathworks.com/hardware-support/robot-operating-system.html].

[7] MUNIN, 2017. Munin/unmanned ship, web page. [Online; accessed 3-January-2017 from http://www.unmanned-ship.org/munin/].

[8] Robotshop, 2017. RPLIDAR. [Online; accessed 3-January-2017 from http://www.robotshop.com/en/rplidar-360-laser-scanner.html].

[9] Santos, J. M., Portugal, D., and Rocha, R. P., 2013. "An evaluation of 2D SLAM techniques available in robot operating system". In 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), IEEE, pp. 1–6.

[10] Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., Klingauf, U., and von Stryk, O., 2014. "Hector open source modules for autonomous mapping and navigation with rescue robots". In *RoboCup 2013: Robot World Cup XVII*. Springer, pp. 624–631.

[11] Muhammad, N., Fofi, D., and Ainouz, S., 2009. "Current state of the art of vision based SLAM". In IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, pp. 72510F–72510F.

[12] Fossen, T. I., 2011. *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd.

[13] Yamauchi, B., 1997. "A frontier-based approach for autonomous exploration". In Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on, IEEE, pp. 146–151.

[14] Wikipedia, 2017. A* search algorithm — wikipedia, the free encyclopedia. [Online; accessed 3-January-2017].

[15] Ferguson, M., 2017. RPLIDAR, source code. [Online; accessed 3-January-2017 from http://wiki.ros.org/rplidar].

[16] Kohlbrecher, S., 2017. Hector SLAM, source code. [Online; accessed 3-January-2017 from http://wiki.ros.org/hector_slam].

[17] Ueland, E. Marine autonomous exploration. [Online; accessed 3-January-2017 from https://www.youtube.com/watch?v=BUihBGbhfDA].