Introducing a Scalable and Modular Control Framework for Low-cost Monocular Robots in Hazardous Environments

Citation for published version:

Digital Object Identifier (DOI):
10.1109/IROS40897.2019.8967521

Link:
Link to publication record in Heriot-Watt Research Portal

Document Version:
Peer reviewed version

Published In:

Publisher Rights Statement:
© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

General rights
Copyright for the publications made accessible via Heriot-Watt Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
Heriot-Watt University has made every reasonable effort to ensure that the content in Heriot-Watt Research Portal complies with UK legislation. If you believe that the public display of this file breaches copyright please contact open.access@hw.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Abstract—Robotics for hazardous environments is currently an important area of research, with the ambition of reducing human risk in potentially devastating situations. Here, we are presenting a Modular Control Framework (MCF) for a low-cost robot with limited sensory resources to address this issue. As a proof of concept, we emulate 3 scenarios — (1) adaptive planning for obstruction avoidance (road block), (2) object identification and support-case-based behaviour adjustment (search and rescue) and (3) autonomous navigation through the environment with reporting of structural status (patrol and monitoring). These were implemented and validated using a Cozmo robot in a small-scale Lego environment. We found that our system can reroute in 90%, can help an injured person 80% and report about failing equipment in 80% of all tested cases, where most of the fails were caused by the object detection used. Our MCF is implemented using ROS, making it easy to use and adjust for other robotic platforms.

I. INTRODUCTION

As the application of robotics is becoming more prevalent in society, many industries are now considering the deployment of robots into hazardous environments, reducing human risk. Space, oil and gas, nuclear and construction industries, to name a few, are all faced with potentially dangerous situations where extra support from technological innovations could be valuable. Such systems can increase efficiency and productivity, as well as reduce unnecessary human exposure to conditions such as fumes, hazardous chemicals and potentially fatal circumstances. This is currently a key area of research, with many governments, institutions and companies around the world investing in the development of such robotic systems. Inspection, maintenance and repair are among the key areas in which robotics can contribute. However, for autonomous mobile robots, even simple tasks such as patrolling sites requires multiple components such as mapping e.g. [1], navigation e.g. [2], obstacle avoidance e.g. [3], etc. These components supply the robot with some form of perception and awareness of the environment.

If an intelligent system is to be deployed in collaboration with or in place of a human, it is important that the system is constructed to effectively interpret and act upon knowledge gained from the environment. To this end, we introduce a scalable and modular framework for robot control using the Robot Operating System (ROS) which addresses the issues mentioned above.

While there has been a huge effort from the community to build such a system, we focus on an architecture that is based on a low-cost consumer platform with limited sensing capabilities, i.e. Cozmo [4] (see Fig. 1). This aims to overcome two important obstacles: i) realistic disaster sites are complex to simulate, ii) the robots currently used in these situations are expensive state-of-the-art research platforms. The contribution of our work is the use of cheap and readily available hardware solutions to overcome issue (ii), therefore only relying on very limited sensing capabilities. Furthermore, using small scale low-cost robots allows us to overcome issue (i) by building cheap and simple scenarios using consumer products such as Lego™. As an additional contribution, we have ensured that the resulting system presented here can easily be transferred to larger, more sophisticated ROS enabled robots and can be scaled to any commonly anticipated scenario, by simply replacing single components used here as a proof-of-concept, with more state-of-the-art solutions. This makes the developed framework applicable for our simulated case but also in real scenarios at future deployment sites. The presented MCF is freely available on GitHub1.

II. BACKGROUND

A. Perception

1) SLAM: Visual perception algorithms such as Simultaneous Localisation and Mapping (SLAM), are implemented with the use of cameras, laser-range finders, and other hardware. A popular SLAM technique is GMapping, a ‘highly efficient’ Rao-Blackwellized particle filter for producing grid maps from laser range data [1], [5]. This is the ‘default’ for most mobile robots [6]. Parallel tracking and mapping (PTAM) [7], is a keyframe based SLAM for stereo systems.

1https://github.com/hmt2/lowcostrobot_mcf
The system and the hardware available. Substituted if necessary, depending on the requirements of the design of our MCF, ORB-SLAM could easily be utilised in our system as it can be considered state-of-the-art. SLAM algorithms such as LSD-SLAM [14], we use ORB-SLAM2 [13] for monocular, stereo and RGB-D cameras. Although we are aware of other monocular features must be extracted from the visual source. SIFT [8], SURF [9] and ORB [10] continue to be popular algorithms for feature detection and are actively used for SLAM. Wang et al. [11] developed an algorithm for moving object detection using monocular vision, improving the robustness of both the state estimation and the mapping processes of SLAM. ORB-SLAM [12] based on the ORB [10] algorithm implements a survival of the fittest based strategy to select points and key frames, which is suggested to be robust. Further to this, R. Mur-Artal et al. extended their algorithm to produce ORB-SLAM2 [13] for monocular, stereo and RGB-D cameras. Although we are aware of other monocular SLAM algorithms such as LSD-SLAM [14], we use ORB-SLAM in our system as it can be considered state-of-the-art. Due to the design of our MCF, ORB-SLAM could easily be substituted if necessary, depending on the requirements of the system and the hardware available.

2) Object Detection: As this paper is not focusing on object detection, we implement a proof of concept for this portion of the proposed architecture using AR markers from the ARToolKit [15]. These AR markers are used in our MCF as a simulation of object detection on more powerful hardware (i.e. with better/more sensors, GPUs, etc).

B. Navigation

Navigation for mobile robots is usually divided into visual and lidar-based navigation. Currently, visual navigation is gaining popularity due to the advent of autonomous vehicles. Approaches such as experience-based robust localisation [16], represents the state-of-the-art in this field.

The most common approach to navigation is using a grid map [2], [17]. These are used to navigate to a given goal in the environment using Dijkstra's algorithm or A* for global path finding, and the dynamic window approach for local path planning and dynamic obstacle avoidance [3]. However, navigating more complex environments and/or longer routes can be a challenge for grid-based navigation due to the size of the search space. To both counteract and associate high-level behaviours with certain locations in the environment, topological navigation is used which can either be hand-crafted or automatically learned, e.g. [18]. Recent approaches aim to modernise this type of navigation by making it robust to changes in the environment [19] showing that it is still state-of-the-art in robotic navigation. One of the earlier approaches to visual navigation [20] uses an omni-directional camera to take pictures at certain locations, i.e. nodes in a topological map, and in-between these nodes.

The ROS navigation stack [17], which is widely used in the robotics community, uses AMCL (adaptive monte-carlo localisation). The purpose of the particle filter is to approximate the posterior of a sample states or particles [21]. Particle filters are successful in low dimensional spaces, and therefore provide assistance for the localisation of robots.

Other examples for visual navigation approaches include using video streams and gradient orientation histograms [22], on-line reconstruction and recognition of 3D objects [23], and identification and localisation of landmarks in images [24]. However, while visual navigation is less frequently used for wheeled indoor robots nowadays, it is often the only form of navigation for Unmanned Aerial Vehicles (UAVs) such as [25], [26].

III. SYSTEM ARCHITECTURE

The following system has been implemented with the limitations of Cozmo in mind. Hence, it relies only on a single sensor, an RGB camera. This, on the one hand, limits the approach in terms of using multi-modal input but, on the other hand, allows to use the system on any robot currently available. The components listed below, rely on the freely available ROS driver for Cozmo [27]. This driver, apart from being able to control the robots motors, provides the odometry, the RGB image, and a ROS tf tree [17].

The complete system architecture can be seen in Fig. 2. The pre-processing module contains the necessary steps required for manipulation of the camera data; raw image processing via filtering and rectification. The perception module consists of the necessary components to form a world model for the agent. This includes SLAM, object detection, and odometry. The navigation module uses perception as an input, allowing for exploration and execution of tasks. The main source of interaction with the environment (see Fig. 3) comes from the planner module, which consists of both the behaviour and actuation. This is where the previous modules will be combined to produce a complete system, capable of executing tasks and behaviour.

A. Perception

1) SLAM: In order for the system to be able to perform a task, it first has to reach its location. To accomplish this, odometry is often used to maintain knowledge of the
current location, however this is subject to errors, which are propagated from one time-step to the next, causing inconsistencies. This is particularly common in tracked vehicles where slippage and friction are more prominent. Due to the downfalls associated with odometry, the only viable sensor for the verification of the robot’s location is the camera.

A popular state-of-the-art localisation method for monocular cameras is ORB-SLAM [12], [13], which has been adopted for this project. ORB-SLAM is an out-of-the-box state-of-the-art SLAM algorithm, especially for monocular systems with a limited field of view.

2) Object Detection: As mentioned in Section II-A.2, AR markers [15] are used as a simulation of object detection on more powerful hardware (i.e. with better/more sensors, GPUs, etc).

First, the camera is calibrated and the image is rectified (pre-processing box in Fig. 2). When a marker is detected in the rectified image, it is compared to the loaded object data. If a match is found, a confidence level is calculated. The translation vector and rotation matrix of each marker, compared to the origin of the camera tf frame, is then computed before being converted into a tf frame itself. The detected marker and the transform between the camera and marker are then published. Due to the modular nature of ROS and the design of the proposed system, the presented module can, at any time, be replaced with a more sophisticated approach that provides the same information.

B. Navigation

ORB-SLAM produces a 3D map, see Fig 4. We disregard the 3rd dimension for navigation and path planning since we are using a wheeled robot. However, due to the limitations of the monocular system, 2D mapping presents several issues, such as missing scale information and loss of tracking when turning corners due to a limited field of view. To overcome some of these issues, we implement a Monte-Carlo Localisation (MCL) that relies on the information provided by ORB-SLAM and the odometry of the robot to create a cloud of particles. This localisation is the input into the global planner developed for Cozmo. Since we do not have costmaps or a grid map as provided in the ROS navigation stack, we use simple topological navigation following straight lines, turning only on waypoints. Hence, all navigation is executed in the global planner. However, recovery behaviours are still possible; due to the use of a topological map, edges can be adapted, i.e. marked as impassable, which is described as part of the behaviours.

1) Monte-Carlo Localisation: Although ORB-SLAM provides a good foundation for navigation, the robot can lose localisation when turning corners or driving too close to a building and therefore navigation is adversely affected. To counteract this, we implemented a custom particle filter to track the current position of the robot, which updates based on the odometry and camera pose, as calculated by the robots’ wheel encoders and ORB-SLAM respectively.

First, the particles are initialised with a position in the map once the first message from ORB-SLAM is published. Once the message is received, a Normal distribution in the x and y directions is created with \( \sigma = 0.1m \), and particles are randomly sampled. A 1D Gaussian distribution is also created for the rotation around the z-axis (\( \theta \)) with \( \sigma = 0.2\text{rad} \) and new values are sampled.

In order to be able to predict where the robot will move in-between receiving the message from the odometry and ORB-SLAM, we use a constant velocity model. Whenever a new odometry message is received, the velocity is updated based on the \( \Delta \) of the current and previous position according to the odometry. The main prediction loop runs at 30hz (3 times faster than the odometry and ORB-SLAM) and uses the constant velocity model to predict the movement of the particles between messages. This increased publishing rate allows for faster position updates and supports a more reactive navigation.

Whenever a new ORB-SLAM pose is received, a list of weights for the particles is calculated based on the distance of the particle to the observed ORB-SLAM pose. A Normal distribution is used to create the values for the weights, biasing them to be clustered closer to the observed pose with \( \sigma = 0.1m \) for \( x, y \) and \( \sigma = 0.2\text{rad} \) for \( \theta \). Once the weights are updated, new particles are drawn with replacement based on those weights. To prevent particle “starvation”, a so-called starvation factor of 80% is used, which defines the percentage of particles drawn based on the calculated weights. The remainder of the particles are drawn randomly based on the current position again, using \( \sigma = 0.1m \) for \( x, y \) and \( \sigma = 0.2\text{rad} \) for \( \theta \). Apart from making the localisation more robust, the addition of the MCL enables the traversal of
unmapped edges.

2) Path Planning: We constructed two ROS action servers [17] for navigation: one for waypoint navigation and one for topological navigation. The tf frame published by the MCL is used to find the angle between the forward pointing $x$-axis of the robot frame and the target waypoint using $\text{atan2}$. This angle, $(-\theta \cdot 0.6)$, is used as the angular velocity to orient the robot towards its next goal. A constant linear velocity of 0.03 m/s is set, until the waypoint is reached. Using 60% of the angle as the angular velocity ensures localisation is not affected. If the robot turns a corner too quickly, ORB-SLAM loses detection of features and therefore, localisation is lost. This percentage was found via trial and error. The actual control of the robots is handled by the Cozmo driver which is provided with a ROS twist message containing the described angular and linear velocities.

We manually composed a topological map by placing nodes (or waypoints) at intersections. The waypoints were created by saving the co-ordinate locations provided by the MCL, connecting them manually via directed edges. These are saved in a yaml file, for the topological navigation server. Fig. 5 shows the constructed waypoint layout and the corresponding edges and names.

Fig. 5. The layout and names of the waypoints for the target environment.

Similar to the global planner in the ROS navigation stack [17], we used Dijkstra’s shortest path algorithm for path planning. This finds the shortest path between nodes in a graph and is therefore used to find the shortest path between the current location of the robot and the goal waypoint. A* search is arguably the more efficient search algorithm, however due to the distance, and therefore equal weight between nodes, Dijkstra’s shortest path works just as well but with less computation. Once the path is calculated, the topological navigation node creates a ROS action client [17] for the waypoint navigation server. The client is called for every step of the path and waits until completion, i.e. until the robot arrives at the given waypoint. This process ends when the final goal is reached.

C. Behaviour generation

To generate the robots’ behaviours, we implemented action servers that have the robot patrol a given route, looking for specific scenarios. These servers used the navigation system, as described above, to fulfill tasks commonly encountered in hazardous environments. The scenarios are described in more detail in Sec. IV.

IV. VALIDATION

To show that the system works in our simulated environment, we implemented three commonly encountered scenarios. Each scenario was specifically chosen to be relevant to the proposed problem of deploying robots into hazardous environments.

The fundamental objective of each goal is to: navigate, report, execute appropriate actions, monitor/report.

Fig. 6. Road block scenario

Scenario 1 - Road Block: The first task for the agent is to modify the route based on road blocks. In lieu of obstacle detection using a laser scanner or similar, the road block is indicated with an AR marker. If this marker is detected close to the next waypoint in the path, the edge is assumed to be non-traversable (see Fig. 6). If a road block is seen and the edge is assumed non-traversable, the agent must plan a new route and execute it. Since monocular navigation does not allow dynamic obstacle avoidance and local path planning, the only way to reach the goal is to change the global path. In order to achieve this, the topological navigation was modified to detect these blockages and remove non-traversable edges. Once the edge is removed, Dijkstra’s algorithm is run to find an alternative path. This is repeated until the goal is reached.

Fig. 7. Search and Rescue scenario, where green 1 and 2 represent healthy agents and red 1 and 2 injured ones – Patrol and Monitoring scenario, where green 3 represents functioning and red 3 faulty gauge readings.

Scenario 2 - Search and Rescue: The second task for the experiment is to search and rescue injured ‘personnel’ from the environment (see Fig. 7). If a person is spotted,
the agent should check if they are injured or not. If not, the agent should politely greet the ‘person’ and continue. If the agent discovers an injured person, the agent should ‘call’ the emergency services, and stay at the scene until the emergency services arrive.

Once the injured person is removed from the scene, the agent should continue on with navigation. Again, the detection of people has been substituted with AR markers.

Scenario 3 - Patrol and Monitoring: The third task tested the agents’ ability to patrol the environment, monitoring and checking ‘gauge’ values when passing them (see Fig. 7). If a ‘gauge’ (AR marker) is discovered, the agent will verbally announce the state of the situation determined by a second marker. If any abnormal readings are found, this should be reported when the agent reaches its navigation goal.

A. Results

We conducted several trials for each scenario in the target environment. In the following, we list the resulting success rates for each scenario.

Scenario 1: We repeated the trial ten times, blocking a node on the direct path to the goal. From observation, we noted the following results: Roadblock was successfully detected 90%, successfully calculated the new route 90%, reached the final goal 90%, and reported the issue correctly 90%.

Scenario 2: For this scenario we placed markers in the environment to simulate healthy and injured persons (see Fig. 1). The first 5 trials were conducted with the markers being placed in easy to reach and see locations. The system achieved a success rate of 100% over all metrics. In the second half of this scenario, we placed the markers in less obvious locations which resulted in the following success rates: the alive person was seen and the correct behaviour was executed 40%, the injured person was seen and the correct behaviour was executed 80%. In all occasions the robot reached its final goal successfully.

Scenario 3: For this scenario, two gauges were placed in the environment, one of them (the second) being faulty. We again tested this behaviour 10 times and noted the following success rates from observation: Seen first gauge and executed correct behaviour 90%, seen second gauge 80% and executed correct behaviour 70%, reached the goal and reported what it had seen 80%, false detections of gauge 20%.

V. DISCUSSION

Our results show that the system is able to successfully navigate the environment and fulfill its given task despite the low-cost hardware and comparatively bad sensor(s). In the following, we will discuss some of the issues that arise from such a system and how we solved them, followed by an analysis of the results from our validation trials.

Mapping: Even when using a state-of-the-art SLAM approach, mapping the environment presents an extensive and time-consuming process. This is mainly due to the limited field-of-view of the camera, leading to loss of tracking when navigating around corners. Additionally, using a monocular camera doesn’t allow the inference of scale from the images, therefore the generated maps do not correctly correspond to the environment. We solved this by using a scaling factor both in the x and y direction to transform from map coordinates to real-world coordinates.

Navigation: By not relying solely on ORB-SLAM for localisation but using our Monte-Carlo Localisation, we were able to compensate for situations where ORB-SLAM lost tracking when going around corners (as mentioned above) and when traversing un-mapped edges.

The main limitation with regards to topological navigation is not using a proper planner because of the lack of laser information and a grid map. Because of this, the waypoints were required to be connected on a straight line. Thus, it is impossible to do dynamic obstacle avoidance. This stems from the limitation of monocular SLAM systems and visual navigation. This has been mitigated by using the global planner to avoid road blocks and choosing a different path. If the system were to be used on a robot equipped with a laser scanner, the navigation could easily be replaced.

A. Scenarios

Scenario 1: For the adaptive navigation, we achieved a 90% success rate, where in one occasion the AR marker was not detected, and therefore no re-planning was carried out. However, Cozmo still reached its goal by simply pushing the marker, which almost completely blocked its vision. Thanks to MCL and the robustness of ORB-SLAM, it was still able to navigate towards its goal. The navigation failed to reach the goal on one occasion, despite it successfully detecting the marker and re-planning. This was due to a localisation failure. In general, these results show that this kind of obstacle avoidance via changing the global plan to re-route the robot is a successful strategy to avoid impassable areas and road blocks.

Scenario 2: It can be seen that if the markers are placed in locations that they can be easily detected, the implemented approach has a perfect success rate. This shows that it works well in principle. However, if the markers are placed in locations that might only be seen while the robot turns, i.e. subject to motion blur or only visible for short times, the approach fails due to not detecting the marker. This could be remedied by using a more advanced object detection approach.

Scenario 3: Similar to scenario 2, if the markers were detected, the behaviour was as expected, except for two occasions where the status of the gauges was not reported. This was due to the robot not reaching its goal location, at which it was supposed to give the report. The reason for this were navigation errors. On all other occasions where it failed, these were due to either false negatives or false positives in the marker detection. On one occasion, a road block was falsely detected, and therefore the robot re-routed and did not see the markers.
In summary, we have seen that our system is able to solve the given tasks with high success rates despite the hardware limitations.

VI. CONCLUSION

This project intended to design and execute a system architecture on a small-scale robotic platform as a proof of concept, assisting in the employment of robotics for hazardous environments. An architecture was designed and implemented for application on Cozmo using ROS. Appropriate methods were adopted; the state-of-the-art monocular SLAM, ORB-SLAM was integrated, topological navigation with Monte-Carlo Localisation was implemented, and AR marker object detection was combined to facilitate goal generation and execution for three different environmental scenarios.

The results of the validation show that the system is capable of adaptive planning for obstruction avoidance, and allows for case based behaviour adjustment and autonomous navigation through the environment, reporting on its status. Moreover, given the fact that all implementation was carried out using ROS, it can be easily transferred onto other robots and extended to more complex scenarios and applications. The fact that the system only relies on a single camera as a sensor further supports the fact that it can be deployed on almost any robot available nowadays.

Future work would extend to the design and integration of reasoning and learning. For example, if the agent is faced with multiple issues at once, it would be beneficial to have some form of decision making for prioritisation of tasks. This will also be tested in larger scale applications and also using different robot hardware. In addition to the research focus of this work, we have already used this framework as a basis for teaching students how to use state-of-the-art software tools in robotics. Due to the low price of the robots used in the work presented, the students can freely experiment with their software building on our framework. We are currently planning a publication describing this approach.

As discussed previously, the main limitation to monocular SLAM is the lack of sensory information for dynamic object avoidance and therefore it would be beneficial to investigate possible approaches to combat this. Another possible development for ORB-SLAM would be to incorporate the odometry of the robot to counteract the scaling problem when using monocular vision.

VII. ACKNOWLEDGEMENTS

This research has been partially supported by ORCA Hub EPSRC(EP/R026173/1) and consortium partners.

REFERENCES