An Improved Robot for Bridge Inspection

H.Peel 1[a], S.Luo [a], A. G. Cohn [b] and R. Fuentes 2[a]

[a] School of Civil Engineering, University of Leeds, United Kingdom
[b] School of Computing, University of Leeds, United Kingdom
E-mail: H.Peel@leeds.ac.uk, R.Fuentes@leeds.ac.uk [Board of Directors]

Abstract – This paper presents a significant improvement from the previous submission by the same authors at ISARC 2016. The robot is now equipped with low-cost cameras and a 2D laser scanner, which is used to monitor and survey a bridge bearing. The robot is capable of localising by combining data from a pre-surveyed 3D model of the space with real-time data collection in-situ. Autonomous navigation is also performed using the 2D laser scanner in a mapped environment.
The Robot Operating System (ROS) framework is used to integrate data collection and communication for navigation.

Keywords – Bridge inspection, monitoring, SfM, SLAM, ROS.

1 - Introduction

Continuing on from work by the same authors [1], more off-the-shelf and low-cost solutions are considered for autonomously navigating a bridge abutment, with the aim of performing a visual inspection on a bridge bearing.

Visual inspection is an important part of inspecting a bridge bearing. In fact, regular inspection is defined in the European and British standard for inspection and maintenance of structural bearings as: "close visual inspection without measurements, spaced at equal reasonably frequent, intervals" [2].

Most of the main problems affecting bearings are reflected in changes to geometry, including: translation, rotation or deformation [3], [4]. Current methods to measure changes in the bearing geometry include [3]: metric tapes, gap gauges, air bubble levels, quadrant rulers, compasses and verniers, levelling and topographic surveys or direct visual observations.

However, regular inspection of bridge bearings often does not occur as frequently as required, in some cases due to difficult access or dangerous conditions. Bridge bearings are critical for the performance in the bridge and inadequate inspection may lead to much greater problems later on in the bridges life.

One solution to increase frequency of inspection is to automate the inspection process. However, the wide range of bridge design and function means that there is not a one size fits all robot for bridge inspection, with technologies being developed for drones [5], underwater vehicles [6] and climbing robots for steel structure bridges [7]. Our contribution is a low cost solution to autonomously performing visual inspection, with technology that can be obtained and implemented in bridge bearing inspection in the near future. In this paper we focus on the implementation of autonomous navigation for autonomous inspection.

Another motive for using robots for inspection is to increase the repeatability of inspections. Previously, we implemented the 3D reconstruction method Structure from Motion (SfM) to enhance the information about a bridge in a format that can be compared directly over time. Now we look at other ways of using this information. Specifically we use a method for Simultaneous Localisation and Mapping (SLAM) in the bridge abutment, where SLAM images can also be used for SfM and visa-versa.

We also consider a second SLAM approach called Hector SLAM that uses LIDAR only and we implement autonomous navigation using a known map and consider some of the challenges of operating in an inspection environment.

2 - Structure-from-Motion (SfM)

Structure-from-Motion (SfM) was used in this work and in previous work [1] as a method for adding value to visual inspection. SfM uses multiple 2D image views to find the 3D geometry (i.e., the structure) of a scene or an object by taking images from different viewpoints (i.e., the camera has motion). The 3D reconstruction software Zephyr Aerial, produced by the company 3Dflow [8], was used for Structure-from-Motion and Multiview stereo calculations and reconstructions in this work. Since SfM is not the primary focus of the work, for a detailed overview of the methods behind SfM and MVS refer to [9] and [10].

3 – Simultaneous Localisation and Mapping (SLAM)

Simultaneous Localisation and Mapping (SLAM) is the process of using sensor readings (e.g., LIDAR, camera and RGB-D sensors) to create a map of the environment whilst at the same time finding the location of the sensor in relation to the map that is being created. There are many implementations of SLAM including filter based methods (such as Extended Kalman filter SLAM [11]), particle based methods (such as Monte-Carlo Localisation [12]) and graph-based methods (such as GraphSLAM [13]).

In this paper, two options for SLAM are considered: ORB-SLAM [14] and Hector SLAM [15]. Both of these methods are performed online - i.e., the computation is done at the same time as the sensor data is being collected. However, due to the computational requirements of these methods, the SLAM calculations were not done on the Raspberry Pi. Instead, The Robot Operating System (ROS) was used to pass camera and laser scan messages respectively to a laptop, where the results are computed.

3.1 ORB-SLAM

ORB-SLAM is a form of Visual SLAM with both monocular and stereo implementations [14]. Features are extracted from images using Oriented Fast and Rotated Brief (ORB) descriptors, chosen for fast extraction and matching overhead compared to other image features, to allow real-time computation without a GPU [14]. The same features are used for tracking, local mapping and localisation for efficiency and reliability [14].

Map points contain information about its 3D position relative to the world coordinate, the direction the point was viewed from, a representative ORB descriptor and the maximum and minimum distances at which the point can be observed.

Before ORB SLAM begins to create a map, a process of initialisation must first occur, with the goal of computing the relative poses between two frames to triangulate an initial set of map points. Only when it is certain that the two views provided will avoid a corrupted map can initialisation be completed, since ambiguity causes all the points to be plotted on a plane [14].

The software implementation of ORB-SLAM used in this work is ORB-SLAM2 [16], which also has a ROS node. ORB-SLAM was chosen as a candidate for localisation because it has been shown to work in urban implementations [14], and has also been implemented using the Raspberry Pi camera in an indoor office environment [17]. Hence, only the low cost camera was required and for localisation and visual inspection.

3.2 Hector SLAM

Hector SLAM was developed for autonomous navigation for urban search and rescue robots [15]. Hector SLAM does not require wheel odometry and relies only on fast LIDAR scan matching [18].

hector_slam [19] is a ROS metapackage that provides packages such as hector_mapping, the ROS node used for SLAM, hector_geotiff which can be used to save the robot trajectory, map and objects of interest in geotiff format. Hector SLAM is designed to be used in 3D, e.g., for robots travelling over rough terrain or aerial vehicles [15], where robots are required to move with up to 6 degrees of freedom. The SLAM system, hector_mapping, is 2D and 3D navigation is achieved by fusing information from an inertial measurement unit. The 2D and 3D solutions are updated separately, but remain coupled in time. Other sensors can also be integrated to decrease uncertainty caused by sensor drift [15], although none are implemented in this paper.

The 2D map is created on an occupancy grid, with interpolation to allow sub-grid accuracy. This approach utilises the high scan rates of modern LIDAR sensors, and provides a more accurate alternative to traditional odometry [18]. Scan alignment is performed based on optimising the alignment of the laser beam endpoints using Gauss-Newton optimisation approach to find the best alignment of the current laser scan data with the existing map through a rigid transform for some cost function [18].

4 - Adaptive Monte-Carlo Localisation

Adaptive Monte-Carlo Localisation (AMCL) is a particle filter method for localisation. Particle filters represent the knowledge a robot has about its position in a given map using a set of particles. Initially, this set of particles is spread over the known map. Measurement and motion models are applied to all of the particles to update the position of each particle. Weightings are then applied to the set of particles depending on the likelihood that a sensor reading at a given location matches the position of a particular particle.

The efficiency of particle filter methods rely on the number of particles being used. The KLD Monte- Carlo algorithm is derived from Kullback-Leibler divergence that adaptively updates the number of particles over time [12], allowing a high number of particles in the initial stages of localisation and a much lower number of particles for tracking when the robot location is known. A detailed description of the AMCL algorithm is available in [12].

5- Platform Description and Integration

5.1 - Platform

The robotic platform, also used in [1] is a DiddyBorg robot [20], a commercially available platform that is built around the Raspberry Pi single board computer, the on-board computer in this work. The DiddyBorg is a six wheeled platform with each wheel powered by a 5V motor. Two expansion boards, produced by the same company, the PicoBorg reverse and the BattBorg, are connected to a Raspberry Pi 2B, which interface with the motors and power the Raspberry Pi using a battery pack, respectively. Some of the motivating factors behind using this platform were its cost, its size and off-the-shelf integration with existing technologies.

In previous work [1], the Raspberry Pi was running Raspbian Jessie, a commonly used operating system for the Raspberry Pi. In this paper, the operating system was changed to Ubuntu MATE 16.04, using the following installation instructions for the Raspberry Pi models 2B or 3 [21]. The main motive for changing operating systems was to be able to access software, such as the Robot Operating System (ROS), more readily, with Ubuntu Mate 16.04 being recommended as the faster and easier way to use ROS on Raspberry Pi. ROS Kinetic was used in this work.

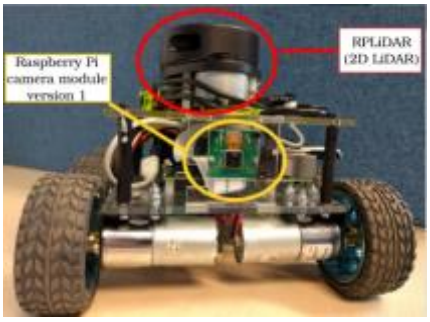

Figure 1: A photo of the DiddyBorg robotic platform with the locations of the RPLIDAR and Raspberry Pi camera sensors used in this work.

5.2 - The Robot Operating System (ROS)

ROS is an open source meta operating system for robots [22], with a large on-line, open-source community. Software is available as packages or stacks that can be easily distributed and shared and developed in multiple languages to allow code reuse in robotics research and development [23]. The software is usually created as independent programs called nodes [24]. Nodes communicate by connecting to a master service and by sending messages that are organised into named topics. Nodes can send information by publishing messages on a topic and other nodes can listen for and subscribe to messages coming from topics. There are defined message types that can be used for specific purposes such as lasers scan messages, camera messages and geometry messages for navigation.

5.2.1 - Motion command node

Scripts to allow navigation of the DiddyBorg mobile platform are based on the original scripts written by the manufacturers [25]. These scripts include python library to interface the motors with the Raspberry Pi through the Picoborg reverse board and the I2C connections on the Raspberry Pi. Using this library to interface with the motors, a ROS node was written that subscribes to a motion command topic in the form of a geometry message with type Twist(), commonly used for velocity messages [26]. These messages are converted into the correct format as used in the aforementioned Python library and the relevant velocity commands are then sent to the Picoborg expansion board. The motor control node is agnostic to the source of the message; hence, this node can be used both for tele-operating the platform and for motion commands for autonomous motion. An example of this process for tele-operating the robot and collecting image data is given in Figure 2.

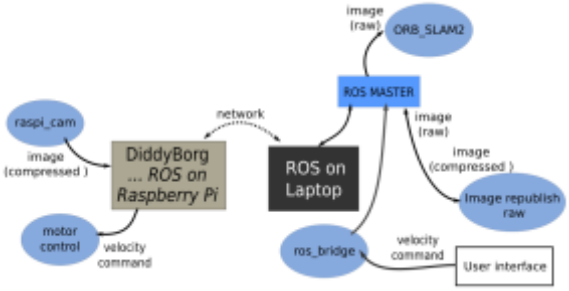

Figure 2: An example of the relationships of different nodes. The route the velocity commands and image data takes between the robot and the relevant node is also shown.

5.2.1 - User interface

In previous work, Node.js was used to create a user interface by incorporating libraries for the Picoborg reverse and the Raspberry Pi camera. This user-interface has now been developed to incorporate Roslibjs [27]. Roslibjs is part of the Robot Web Tools effort and is a JavaScript library for interacting with ROS from a web browser. Roslibjs allows the functionality of ROS such as publishing and subscribing to messages, service calls and other core ROS functionality. This user interface allows the teleoperation of the robot from a web-browser and can be accessed from a mobile phone.

6- Other Hardware and Sensors

6.1.1 Raspberry Pi camera module

Raspberry Pi camera module version 1 was used for photographic data collection. This camera is capable of producing 5MP pictures and 1080p HD video at 30fps. The camera module is not automatically compatible with a Raspberry Pi 2B running Ubuntu Mate, hence some adjustments and drivers were required, as described in [28]. A ROS package, raspicam_node has also been developed for the Raspberry Pi camera by [28].

6.1.2 - Camera calibration

Camera calibration is required for ORB-SLAM. The same calibration parameters were used as an input to SfM. However, the software is capable of finding the calibration parameters automatically and these parameters were adjusted by the SfM software. The monocular calibration script from ROS [29] was used to calibrate the Raspberry Pi camera. An 8x6 chessboard was placed on a planar surface and the camera was moved to obtain different viewpoints. The calibration software indicates when sufficient samples have been collected to perform a calibration. However, further samples were taken until the software indicated that the samples taken were above a 'good' threshold for translation in x, y and skew (indicated in a traffic light system from red to green). This number of samples translates to around 120 readings, which is comparable to [17].

6.1.3 - RPLIDAR

Previous work [1] considered the requirements of autonomous navigation of robots in inspection environments. Camera data was used for SfM reconstructions, which were then processed to create a 2D map, with the goal of localising in the map using ultrasound sensors.

In this work, mapping and localisation in an inspection environment was also considered, but this time a 2D LiDAR, the RPLIDAR version A1, was used. The RPLIDAR is a low-cost (£300) 2D LIDAR solution developed by RoboPeak. The sensor has a range of 6m in 360° with readings being taken at 5.5Hz. The sensor was easily incorporated into the current setup, with mounting possible directly onto the top of the DiddyBorg platform. The sensor power supply is 5V, with a USB connector which can be powered directly from the Raspberry Pi. Drivers and ROS packages are readily available on the Raspberry Pi and can be installed from [30].

7 - Site Description

The site considered in this paper is the Millennium Bridge, a cable suspension footbridge in Leeds, UK. The bridge, which opened in 1993, crosses the River Aire spanning approximately 57m to connect The Calls to Brewery Place. The bearings on the north side of the river were used as the site for data collection.

The North side bearings are situated in the top abutment, and its dimensions are approximately 2.8x1.2m. There is a trough that runs alongside one side of the site, and there are various pipes and electrical cables running along the length of the enclosure. The top bearings are seated on the bridge by means of a machined steel plate bolted to the bearing.

8 - Survey and Data Collection

Data collection was performed by tele-operating the robot in the bearing enclosure using the user interface described in section 5.2.2. A router was used to allow networking between the DiddyBorg platform and a laptop.

Next, data collection was performed. At this stage, data from the raspicam_node and the rplidar node were recorded into rosbags to allow post-processing of the data. Rosbag is a command-line ROS tool for storing serialised ROS messages in a file as messages are received from specified ROS topics. This tool allows the data to be replayed through the ROS topics at a later date. LiDAR data was collected on two separate instances. One set of data was used to create a map, and the other was used as test data for the localisation algorithm, see section 9.3.

The camera resolution was set at 320x200 pixels to allow real-time processing of the camera data. This resuolution is a similar to the one used in [17]. Camera data was sent in jpeg compressed form by the raspicam node, received and uncompressed using ROS image_transport tools on the laptop and then processed by the ORB-SLAM2 node, as depicted in Figure 2.

As discussed in section 3.1, before mapping can begin, initialisation of ORB-SLAM must occur. Once initialisation occurred, the DiddyBorg platform was navigated around the bearing enclosure to build up a map of the environment. Three repetitions of ORB-SLAM were completed. In contrast to previous work [1], separate data was not collected for the SfM calculations, but the data collected for ORB-SLAM was also used for this purpose.

9 - Results and Discussion

The effectiveness of the SLAM methods ORB-SLAM and Hector SLAM is now considered. A qualitative comparison between the methods is given in Table 1.

9.1 - ORB-SLAM

As described in [31], changes in lighting conditions causes tracking failure between frames for multiple feature descriptors. This phenomenon also affected the localisation and mapping of ORB-SLAM, as expected. Since data collection was performed in the late afternoon, by the third repetition the sun had set below the level of the bridge, see Figure 3. As a result, there was a greater contrast between lighting conditions in successive frames and tracking was lost for a large period of time, see Figure 3.

Figure 3: Two frames from ORB-SLAM that results in the map shown in Figure 4. ORB features are marked as green squares in the left hand image. In the right hand image no features appear since tracking has been lost due to a sudden change in lighting conditions.

Figure 4: The map produced by ORB-SLAM. The key frame locations can be seen by the blue rectangles, the current keyframe as a green rectangle and the progression of the robot also in green. The red map points show the local visible map.

In general, initialisation was obtained quickly, with sufficient features provided by objects in the environment (e.g., railings and litter). Throughout the mapping, adequate features were present, with texture being provided by dirt and cracks on planar surfaces. Figure 3 shows a comparison of the ORB-SLAM results when initialisation is successful and ORB frames are being tracked and some frames later when tracking has been lost due to abrupt change in brightness.

Tracking was also lost due to abrupt and fast motions, as anticipated. When the tracking was lost, the robot has to return to a previous key frame and localisation is performed globally, this was performed successfully in most cases. As a result of loss of tracking the whole area mapped with the same detail, this is likely to affect localisation with new data.

It is also possible to save and reload a map using additional functionality developed by [32]. The localisation mode in ORB-SLAM can then be used with a loaded map and new data for localisation. In future work, the expansion of this tool for autonomous navigation will be considered, where one obstacle to over-come is to provide scale to the map.

9.1.1 - SfM Results

The data collected and used for ORB-SLAM in Figure 4 was also sufficient the SfM reconstruction to be successful. Approximately 230 images were collected in total and used for the SfM reconstruction. SfM was not as affected by the variations in brightness and was able to use more of the dataset, whereas ORB-SLAM cannot use the frames where tracking was lost. Although the front wall of the abutment in Figure 5 was one of the least detailed areas of the SfM reconstruction, it was much more detailed than the corresponding region in Figure 4. Similarly, it can be seen that there are more areas in Figure 5 where photos were used compared to Figure 4, both indicated by blue triangles.

Figure 5: The SfM reconstruction completed using Zephyr Aerial SfM software. The same dataset used for ORB-SLAM was used for the reconstruction. The camera positions can also be seen as blue triangles.

9.2 - Hector SLAM

The environments shown in Figure 5 and Figure 8 show some inconsistencies. In Figure 5 the curved wall at the front of the enclosure can be clearly seen, there is no sign of this wall in Figure 8 . The reason for this discrepancy was the height of the RPLIDAR with regards to the wall – when mounted on the DiddyBorg platform the RPLIDAR was higher than the wall, and hence the wall not detected by the sensor.

The map created by Hector SLAM in Figure 8 shows lines that go off the map. These lines are sensor readings recorded by the LIDAR at maximum sensor

range. The bridge in question crosses the River Aire and these maximum readings are caused by reflections off the water.

In addition, Figure 8 shows errors in the SLAM results that occurred when the robot mounted an electrical cable that was present in the bearing enclosure. This cable caused the base, and hence the RPLIDAR, to tilt into a different plane. As discussed in section 3.2, Hector SLAM can be used in situations where the plane of the LIDAR changes, allowing 3D navigation. To achieve this an IMU is required and integrating this sensor information to track the transformation of the base to a stabilised base [18]. This consideration will be included in future work to give a more robust SLAM system.

To compare the results from ORB-SLAM and hector_slam the trajectory outputted by both methods was recorded simultaneously and plotted in Figure 6. The results show overall the trajectories are very similar in shape. However, the trajectory for ORB-SLAM is less accurate and has greater variation than Hector SLAM and gaps appear when the trajectory crosses itself.

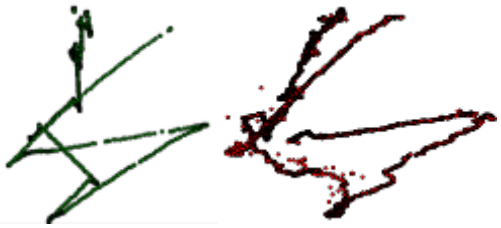

Figure 6: A comparison between the trajectories of the robot using Hector SLAM (left) and ORB-SLAM (right). Note the figures are not plotted together since the ORB-SLAM results require scaling.

10 – Results:Hector SLAM for Autonomous navigation using AMCL.

It is possible to perform autonomous navigation using the hector_navigation stack [33], also developed for urban search and rescue environments by the same authors of Hector SLAM. The hector_navigation stack contains packages such as the hector_exploration_node, which accesses the hector_exploration_planner, a planning library that allows the robot to explore unknown areas of the map. However, the resulting map cannot be saved and used again with the hector_navigation stack.

For inspection applications it is useful to have a known map to highlight targets for inspection or areas of interest, in advance. For this reason, Adaptive Monte Carlo Localisation (AMCL) was used for localisation in a known map; the map was created using hector_slam. As in section 8.2, the hector_slam package was used to provide odometry from the 2D LiDAR data. This process was visualised using rviz and displayed in Figure 9a-c. The implementation used in this work is based on [34].

Initially, the front wall of the enclosure was not registered in the map, but in reality these areas past the front wall are not accessible. For caution, since no method has been implemented here to prevent navigation to areas beyond the front wall, the navigation system was not tested in the abutment enclosure. However, LIDAR data collected from the bridge is used as the input and the same commands for a particular navigation goal were successfully received.

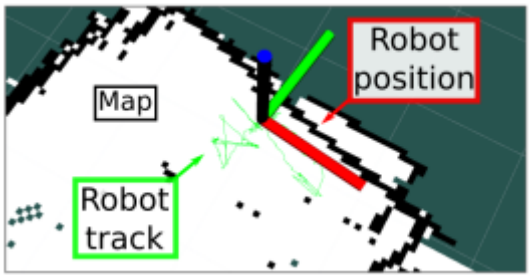

Figure 7**:** Shows the result of hector_slam with the robot position and path marked by in the figure. Localisation and mapping errors occur in this example because the RPLIDAR is tilted out of its original plan without any update e.g., using data from an IMU.

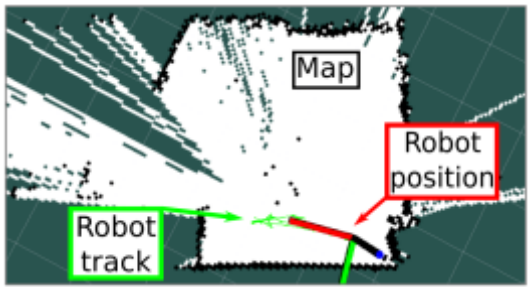

Figure 8: The SLAM result for one set of LIDAR data using Hector SLAM. The robot position, trajectory and map boundaries are shown. Maximum sensor values are also returned in some instances – shown by lines that go outside the map.

Conversely, one of the disadvantages of using pre-existing maps is if the environment changes, the map may no longer be representative. In Figure 9c) the front wall of the abutment was picked up momentarily by the LIDAR. As a result, localisation was temporarily lost, since the wall is not a known landmark and some particles are placed outside of the map.

One disadvantage of AMCL is that a start position is required to be set for the localisation process to begin. Since the geometry of the bridge is well known in this

case, an initial position can be set fairly accurately. This can also be set automatically. Figure 9 shows the scatter of particles before the initial position and again once the particles have converged to a location.

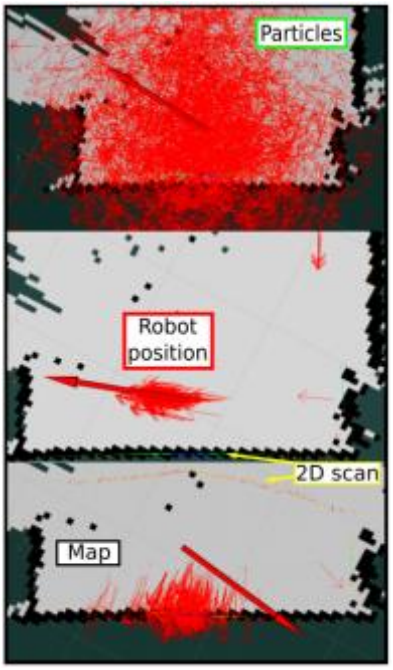

Figure 9: a, b and c top to bottom.
a) Initially, particles are spread across the whole map.
b) After a few time steps particles converge.
c) Change in map – front of the enclosure detected by 2D LiDAR, position estimation of robot was lost.

Table 1: A qualitive comparison between ORB-SLAM and Hector SLAM with AMCL.

	ORB-SLAM (localization only)	**Hector SLAM Plus AMCL**
Cost	Lowest	Highest
Accuracy	Lowest	Highest
Automatic scale	No	Yes
3D map	Yes	No
Use data elsewhere	Yes: use directly for SfM.	Require camera for visual inspection.
Environment variation/ Real world robustness	Affected by lighting. Harder to relocalise	Lost if scan doesn't match environment, but relocalises well.
Autonomous Navigation	Need some method for scaling first.	Easily implemented.
Future work	Sensor Fusion: odometry for scaling	Sensor fusion: IMU for 3D navigation

Table 1 compares some qualitative differences between using Hector and AMCL and ORB-SLAM for localisation. Note, that since no method for navigation is implemented for ORB-SLAM in this paper, localisation only is considered. Overall, the low cost and re-use of data are key advantages of ORB-SLAM, but Hector SLAM with AMCL is more accurate, and autonomous navigation could be implemented by outputting the position in the map and the required target to the ROS navigation stack. Both methods require future work to improve their robustness in inspection environments, but the results so far are promising.

11- Future Work and Considerations

To increase the robustness of the SLAM and autonomous navigation approaches used here, sensor fusion will be implemented, primarily with an inertial measurement unit for the hector_slam approaches and odometry for ORB-SLAM to incorporate scale into the map. Future work will look at extending the ORB-SLAM localisation method considered here for navigation.

12 - Conclusions

In this work, an improved robot for inspection of bridge bearings with off-the-shelf low-cost camera and LIDAR technology was presented. Building upon previous work, existing methods for Simultaneous Localisation and Mapping (SLAM) (i.e., ORB-SLAM and Hector SLAM) were applied to a real bridge. Visual inspection was carried out and qualitatively compared the differences in the methods. Using Hector SLAM, we then explored methods for autonomous navigation, with a known map.

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council (ESPRC), UK.

References

[1] H. Peel, G. Morgan, C. Peel, A. Cohn, and R. Fuentes, "Inspection robot with low cost perception sensing," in ISARC 2016 - 33rd International Symposium on Automation and Robotics in Construction, 2016.

[2] H. Eggert and W. Kauscke, "Structural Bearings," vol. 3, no. November, p. 405, 2002.

[3] L. M. R. Freire, J. De Brito, and J. R. Correia, "Management system for road bridge structural bearings," Structure and Infrastructure Engineering, vol. 10, no. 8. Taylor & Francis, pp. 1068–1086, 2014.

[4] [4] L. Freire, J. de Brito, and J. Correia,

"Inspection Survey of Support Bearings in Road Bridges," J. Perform. Constr. Facil., vol. 29, no. 4, p. 4014098, 2015.

[5] [5] J. Chen, W. Junjie, C. Gang, W. Dong, and X. Shen, "Design and Development of a Multi-rotor Unmanned Aerial Vehicle System for Bridge Inspection," in International Conference on Intelligent Robotics and Applications (ICIRA 2016), 2016, pp. 498–510.

[6] [6] R. R. Murphy et al., "Robot-assisted bridge inspection," J. Intell. Robot. Syst. Theory Appl., vol. 64, no. 1, pp. 77–95, 2011.

[7] [7] N. H. Pham and H. M. La, "Design and implementation of an Autonomous Robot for Steel Bridge Inspection," Int. J. Adv. Robot. Syst., vol. 10, no. January, pp. 556–562, 2016.

[8] 3DFLOW, "3DF Zephyr Aerial." 3DF, Verona, 2016.

[9] R. Toldo, R. Gherardi, M. Farenzena, and A. Fusiello, "Hierarchical structure-and-motion recovery from uncalibrated images," Comput. Vis. Image Underst., vol. 140, pp. 127–143, 2015.

[10] R. Toldo, F. Fantini, L. Giona, S. Fantona, and A. Fusiello, "Accurate Multiview Stereo Reconstruction with Fast Visibility Integration and Tight Disparity Bounding," ISPRS-International Arch. Photogramm. Remote Sens. Spat. Inf. Sci., vol. 1, no. 1, pp. 243–249, 2013.

[11] D. Thrun, Sebastian; Burgard, Wolfram; Fox, "EKF Localization," in Probabalistic Robotics, Cambridge, Massachusetts: The MIT Press, 2006, pp. 54–65.

[12] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots."

[13] D. Thrun, Sebastian; Burgard, Wolfram; Fox, "The GraphSLAM Algorithm," in Probabalistic Robotics, 10th ed., Cambridge, Massachusetts: MIT Press, 2006, pp. 337–385.

[14] J. D. Mur-Artal, Raúl, Montiel, J. M. M. and Tardós, "ORB-SLAM : a Versatile and Accurate Monocular SLAM System," IEEE Trans. Robot., vol. 31, no. 5, pp. 1147--1163, 2015.

[15] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," in Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), 2011, pp. 155–160.

[16] Raulmur, "ORB_SLAM2." raulmur / github repository, 2016.

[17] G. Ponnu, J. George, and J. Skovira, "Real-time ROSberryPi SLAM Robot," Cornell University, 2016.

[18] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, O. Von Stryk, and U. Klingauf, "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots," Rob. 2013 Rob. 2013 Robot World Cup, vol. XVII, pp. 642–631, 2013.

[19] Tu-darmstadt-ros-pkg, "hector_slam." Tu-darmstadt-ros-pkg / github repository, 2016.

[20] PiBorg, "DiddyBorg - The most powerful Raspberry Pi robot available | PiBorg," Drupal. [Online]. Available: https://www.piborg.org/diddyborg. [Accessed: 29-Mar-2017].

[21] Martin Wimpress, "Ubuntu MATE for the Raspberry Pi 2 and Raspberry Pi 3," 2016. [Online]. Available: https://ubuntu-mate.org/raspberry-pi/. [Accessed: 29-Mar-2017].

[22] The Open Source Robotics Foundation, "ROS.org | About ROS." [Online]. Available: http://www.ros.org/about-ros/. [Accessed: 18-Mar-2017].

[23] Open Source Robotics Foundation, "Introduction," ROS Wiki, 2014. [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed: 25-Mar-2017].

[24] J. M. O'Kane, A gentle introduction to ROS. Jason M. O'Kane, 2014.

[25] PiBorg, "PicoBorgRev." 2015.

[26] ROS, "common_msgs - ROS Wiki," Open Source Robotics Foundation, 2014. [Online]. Available: http://wiki.ros.org/common_msgs. [Accessed: 29-Mar-2017].

[27] Robot Web Tools, "roslibjs." Robot Web Tools / github repository, 2017.

[28] larrylisky.com, "Enabling Raspberry Pi Camera V2 under Ubuntu Mate." [Online]. Available: https://larrylisky.com/2016/11/24/enabling-raspberry-pi-camera-v2-under-ubuntu-mate/. [Accessed: 29-Mar-2017].

[29] 38 contributors (full list Https://github.com/ros-perception/image_pipeline/graphs/contributors), "ros-perception / image_pipeline." github repository.

[30] Robopeak, "rplidar_ros." robotpeak / github repository, 2016.

[31] A. Pieropan, M. Björkman, N. Bergström, and D. Kragic, "Feature Descriptors for Tracking by Detection: a Benchmark," 2016.

[32] poine and raulmur (forked from raulmur), "ORB_SLAM2." , poine / raulmur / github repository, 2016.

[33] Tu-darmstadt-ros-pkg, "hector_navigation." Tu-darmstadt-ros-pkg / github repository, 2016.

[34] Pariljain, Nischal92, ParitoshKelkar, Nischalkn, "F1tenth.", nischalkin / github repository, 2016.