University of Nevada, Reno

Control and Navigation Framework for a Hybrid Steel Bridge Inspection Robot

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

by
Hoang Dung Bui

Dr. Hung M. La - Thesis Advisor
May 2021

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

Hoang Dung Bui

entitled

Control and Navigation Framework for a Hybrid Steel Bridge Inspection Robot

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Hung M. La, Ph.D.
Advisor

Alireza Tavakkoli, Ph.D.
Committee Member

Hao Xu, Ph.D.
Graduate School Representative

David W. Zeh, Ph.D., Dean
Graduate School

May, 2021

Abstract

Autonomous navigation of steel bridge inspection robots is essential for proper maintenance. Majority of existing robotic solutions for steel bridge inspection require human intervention to assist in the control and navigation. In this thesis, a control and navigation framework has been proposed for the steel bridge inspection robot developed by the Advanced Robotics and Automation (ARA) to facilitate autonomous real-time navigation and minimize human intervention. The ARA robot is designed to work in two modes: mobile and inch-worm. The robot uses mobile mode when moving on a plane surface and inch-worm mode when jumping from one surface to the other. To allow the ARA robot to switch between mobile and inch-worm modes, a switching controller is developed with 3D point cloud data based. The surface detection algorithm is proposed to allow the robot to check the availability of steel surfaces (plane, area and height) to determine the transformation from mobile mode to inch-worm one. To have the robot to safely navigate and visit all steel members of the bridge, four algorithms are developed to process the data from a depth camera, segment it into clusters, estimate the boundaries, construct a graph representing the structure, generate the shortest inspection path with any starting and ending points, and determine available robot configuration for path planning. Experiments on steel bridge structures setup highlight the effective performance of the algorithms, and the potential to apply to the ARA robot to run on real bridge structures.

Acknowledgments

I would like to thank my advisor, Dr. Hung La, who provided me with the guidance and assistance needed for my research. Without him and his advice, this thesis would not have been possible. I would also like to thank my committee members, Dr. Hao Xu and Dr. Alireza Tavakkoli, for their advice and time taken to review this thesis. Lastly a thank to my wife, Hoa Bui, who is always support me and make me calm in the stressful time.

This work is supported by the U.S. National Science Foundation (NSF) under grants NSF-CAREER: 1846513 and NSF-PFI-TT: 1919127, and the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/OST-R) under Grant No. 69A3551747126 through INSPIRE University Transportation Center (http://inspire-utc.mst.edu) at Missouri University of Science and Technology. The views, opinions, findings and conclusions reflected in this publication are solely those of the authors and do not represent the official policy or position of the NSF and USDOT/OST-R.

Table of Contents

List of Figures

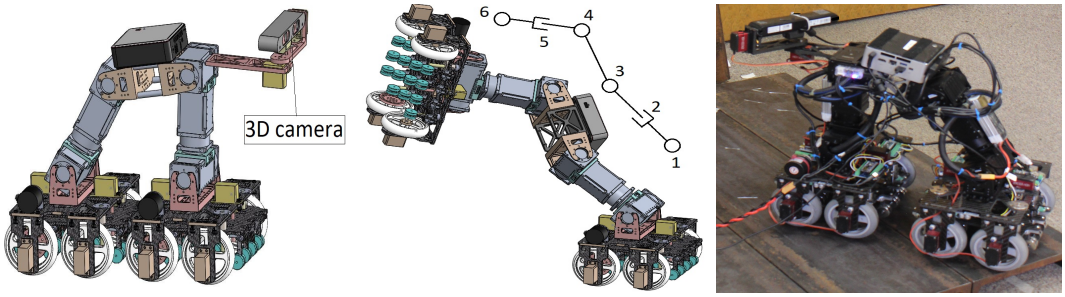
Chapter 1

Introduction

Within the field of health monitoring of bridge structures [2–5], the development of novel robotic platforms has received considerable attention in the recent years [6–8, 8–15]. It has been increasingly stressed in the literature that timely and regular monitoring of steel bridges ensures the safety of transportation vehicles. Environmental degradation (e.g., rain, wind, solar radiation), continuous surface-level friction, overloading, and other factors lead to deterioration of different structures on steel bridges. Continuous steel bridge monitoring is necessary to ensure transportation safety and proper maintenance. The tasks can be done manually, however, it is time-consuming, labor intensive, dangerous, affect to the traffic, and sometimes inaccessible for human in complex structures. For the reasons, there are varieties of robotic platforms [1, 16–19] developed to support human to do the task. These robots are magnetic-based that help them traverse on multiple angles of steel bridge structures. Most of the robots are controlled manually by an operator.

As an effort to go further in the field, the Advanced Robotics and Automation (ARA) Lab of the University of Nevada, Reno has developed a bio-inspired hybrid

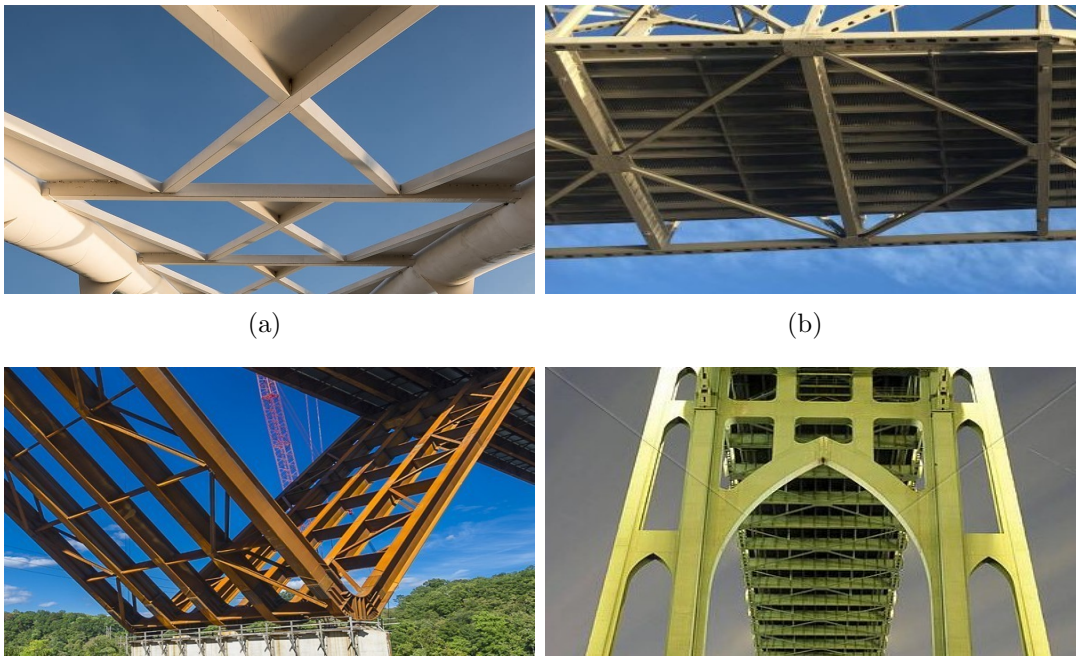
robot - ARA robot [1, 20] (Fig. 1.1) with the aim to inspect a steel bridge structure autonomously. The robot is able to work in two modes: (1) mobile to traverse on smooth steel surface, and (2) inchworm - to change/jump to another steel bar surface. In this thesis, a control and navigation framework is proposed for the ARA Lab's robot to move easily and autonomously on smooth steel surfaces, and jump to another steel surfaces, which are adjacent or higher than the current one.



(a) (b) (c)

Figure 1.1: ARA robot model [1] in (a) *mobile*, (b) *inch-worm modes*, and (c) real robot

To move on smooth steel surfaces, the robot needs to navigate itself on varieties of structures in steel bridge as shown in Fig. 1.2, which consists of popular structures as *Cross*-, *T*-, *I*-, *K*- and *L*- shape. The structure's complexity and varied dimensions make motion planning task is very challenging, requires the robot's perception about the structure and a method that is able to make use of limited work space. Moreover, to navigate the robot to inspect a bridge continuously, the navigation system needs to build a path in large scale. Combining with a depth sensor to collect data in a far distance, a method is developed to build and represent a steel bridge structure as a graph. Moreover, we propose a variant of open Chinese Postman Problem (VOCPP), which determines the shortest path to inspect all available steel bars on the structure with difference of starting and ending points.



(a) (b)

(c) (d)

Figure 1.2: The typical steel bridges structure: (a) cross shape, (b) K-Shape, (c) L-Shape, (d) and combination of shapes

1.1 Literature Review

Most of steel bridges are monitored by civil inspectors manually [21]. However, due to the complex structural composition and inaccessible regions of the bridges (e.g., pipes, poles, overhead cables), the manual inspection of these regions is a perilous task for human inspectors. Additionally, manual inspection is time-consuming, labor-intensive, and disruptive to traffic. It is for this reason that different robotic solutions have been developed for automated steel bridge inspection [17, 22–29]. These robots are equipped with different adhesion mechanism (e.g., magnetic wheels, pneumatic, suction cups, bio-inspired grippers), visual sensors (e.g., monocular, stereo vision, RGB-D sensors) and other sensory modalities to facilitate navigation and inspection (e.g., IMUs, eddy current sensors) [17, 22, 24–28]. Adhesion force generated by either permanent or electro magnets attached on the robot enables them [29] to adhere and

navigate flexibly on smooth steel surfaces. The incorporation of legged mechanism with electromagnets allows robots to assist in locomotion and traversal through complex steel structures [30]. These robots are designed for a particular environment, lacking the deploy-ability in many unstructured environments. A flexible and versatile climbing robot was designed in [21], which was equipped with 5-DOF arm, eddy current sensor and RGB-D sensors for inspection of steel bridge surfaces, especially for inaccessible regions of the bridges. Another type of climbing robot was developed by [31] with untouched magnet blocks to move efficiently on metal surfaces. Although these robots alleviated the difficulty of moving on complex steel surfaces, they were controlled manually by cables or remote instruction from human operators.

There is a number of work related to the navigation of inspection robots for steel bridges [32–34], which helps the robots move in a local area, and assume the robot dimension quite small comparing to the workspace. In [32] particularly for autonomous steel bridge inspection robot, the authors proposed a task-level primitive and online navigation combining with IR sensor, which helps the robot move in local area. In [33], the authors proposed a method to detect edges on a large surface, which is used in navigation. The method in [34] supported the small size robot to move in a large-inside steel bridge space. Their navigation work here assumed that the robot's dimensions are quite small comparing to the workspace. With the limit of steel bar dimension, our research needs to handle a new circumstance for robot navigation and motion planning.

There is an important feature in navigation for steel bridge inspection robots: the dimensions of steel bars are limited, and the robots have small space to make a motion. The methods in [35–37] are able to build very nice convex regions, which make the construction of configuration space easy. The approximation methods, however, reduce the dimension of the workspace, and could make the robot motion

infeasible. To overcome these problems, in this paper we propose a method, which segments the workspace into multiple clusters and represents it by a set of boundary points. The method can use all the possible area in the workspace, thus increases the probability of finding a path for the robot. With the irregular shape of the steel bar, Expectation Maximization - Gaussian Mixture Model (EM-GMM) method [38–41] is utilized to segment the data into irregular dimensional clusters.

When perceiving the working space, the navigation system represents the bridge structure as a graph. Estimation of features from point cloud data receives a significant attention from researchers [42–44]. These researches worked on a particular small object to the view space of the depth sensors. In our research, the bridge is large and usually over than the sensor view. A graph construction algorithm is developed for this purpose. From the built graph, the next step is to determine a shortest path to inspect all steel bars in the structure, that is called *inspection route* or *Chinese Postman Problem* (CPP). There are several requirements for the shortest path in our context. The real bridge is too long for the depth sensor to collect data in one frame, thus the bridge is separated into multiple parts. As the robot finishes inspection in one part, it will move the next one to continue the task. Therefore, the robot starts at one point and ends in another point. The starting and ending points can be anywhere on the bridge structure, which are convenient for the robot to perform the next task. There is a number of research working on CPP problem [45–47], however, there is no work satisfying our requirement of arbitrary starting and ending locations. Therefore, we propose a variant of open CPP (VOCPP) algorithm to handle the problem.

1.2 Contributions

Steel bridge inspection is a continuous process, the primary goal of our research is to develop a fully autonomous robotic system to automate this task. In this research, we proposed a control and navigation framework for the ARA robot to navigate autonomously on steel bridge structures. The contributions of this thesis are then as follows:

- A control framework to help the ARA robot switch between two operation modes (mobile, inch-worm) autonomously. The switching control determines the availability of the planar surface, its area and height to decide the next transition;
- A non-convex boundary estimation algorithm to find the boundary of the steel bar point cloud, that utilizes the availability of limited steel bar surface.;
- An area estimation algorithm using point cloud data from RGB-D sensor to allow the robot to assess area availability for transitioning from one plane to another. This algorithm determines if the available area is sufficient for the robot's foot transition;
- An efficient algorithm to segment the steel bridge structure into steel bars and cross area regardless any kinds of input structure (tested on *T*-, *K*-, *I*-, *L*- and *Cross*- shape);
- An algorithm to construct an undirected graph from the steel bridge structure data;
- *VOCPP* algorithm to find the shortest path for the robot to inspect all steel bars in the graph with difference between the starting and ending points;

- A method to determine whether a robot configuration belongs to free configuration, with the input as a set of cluster boundary of steel bridge structure.

1.3 Thesis Organization

The organization of the remainder of this thesis is as follows. Chapter 2 provides fundamental background including Point Cloud processing, motion planning, robot control, and Chinese Postman Problem. Chapter 3 introduces the control framework of the ARA robot. Chapter 4 introduces the navigation of the ARA robot in mobile transformation. Chapter 5 presents the experiment setup and results. Lastly, Chapter 6 covers the conclusions with analysis and potential future development.

Chapter 2

Background

This chapter provides the background knowledge, which is used in this thesis such as Expectation Maximization - Gaussian Mixture Model, Chinese Postman Problem, and Rapidly Exploring Random Tree Path Planning.

2.1 Gaussian Mixture Model

Gaussian Mixture Model - GMM [48] is a probabilistic model with an assumption that the data is generated by a group of Gaussian distributions with unknown parameters. Assuming there are n observations $X_1, X_2, ..., X_n$, and each X_i is drawn from one of K mixture components. Going with each random variable X_i, there is a label $Z_i \in \{1, ..., K\}$, which shows the component X_i came from.

From probability law, the marginal probability of X_i is:

$$P(X_i = x) = \sum_{k=1}^{K} P(X_i = x|Z_i = k)P(Z_i = k) = \sum_{k=1}^{K} P(X_i = x|Z_i = k)\pi_k \qquad (2.1)$$

where, π_k is called mixture proportions, and it represents the probability that X_i belongs to the $k-th$ mixture component. The sum of the mixture proportions is one: $\sum_{k=1}^{K} \pi_k = 1$.

2.2 Expectation Maximization GMM

Expectation Maximization (EM) algorithm is an iterative method [49], which determines the best value for a process latent variable in the presence of of latent variables. There are two modes in the EM algorithm: estimating step and maximizing step.

- Estimating step so-called E-step attempts to estimate the value of the missing or latent variables.

$$\gamma_j(x_n) = \frac{\pi_j P(x_n|\mu_j, \sum_j)}{\sum_k \pi_k P(x_n|\mu_k, \sum_k)} \tag{2.2}$$

- From the estimation, the maximizing step or M-step optimizes the model parameters to explain the data.

$$\mu_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n) x_n}{\sum_{n=1}^{N} \gamma_j(x_n)} \tag{2.3}$$

$$\sum_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n)(x_n - \mu_j)(x_n - \mu_j)^T}{\sum_{n=1}^{N} \gamma_j(x_n)} \tag{2.4}$$

$$\pi_j = \frac{1}{N} \sum_{n=1}^{N} \gamma_j(x_n) \tag{2.5}$$

where, γ_j is the weight to component j of variable x_n, μ_j are the means, $\sum_j$ are covariances, and π_j are the mixing probabilities.

For GMM, the model's parameters such as the variances and means are unknown, thus EM algorithm is a good tool to find the parameter values. In our case, the number

of steel bars and cross area are unknown, and need to be determined. Running the EM-GMM algorithm iteratively until it provides a stable result.

2.3 Rapidly Exploration Random Tree

Rapidly Exploration Random Tree (RRT) [50–52] is a path planning algorithm, which works efficiently on nonconvex, high dimensional space. The idea is to build a tree that expands incrementally by random samples from the search space. As each new sample is drawn, it is attempted to connect to the nearest neighbor state in the tree. If the connection is feasible, the sample is added to the tree, and the tree gets new state. The length of the connection is limited to assure all the points between them belonging to the search space. The random sample is a tool to control the direction of the tree expanding, and the length limit regulates its rates. The computation time depends on the number of samples and length limit.

RRT growth's direction can be adjusted by adding the probability of sampling states from a particular area. It will speed up the searching and guide the tree to the planning problem goal. However, RRT does not usually bring an optimal solution, and the path is not smooth.

2.4 Chinese Postman Problems

Chinese Postman Problem (CPP), also known as Route Inspection problem [53], is a problem of graph theory that determines the shortest delivery path for a mailman with two criteria: (1) the mailman starts and ends at the same point, and (2) he needs to traverse all the streets at least one. If all corners in the graph are even degree, an ideal solution will exist, and all the street/edges are traversed only one.

If the starting point and ending points are different, the problem is called *Open Chinese Postman Problem*. If only the starting and ending points are odd degree, then it exists an ideal path, which goes from the starting point to the ending one, and traverses all the edges only one. The condition to solve CPP and Open CPP will be used to develop the Variant Open CPP algorithm in chapter 4.

Chapter 3

Control Framework

ARA robot can configure itself into two transformations: mobile and inch-worm. In this work, we integrated a switching control mechanism (shown in Fig. 3.1) to the robot [20]. This control mechanism enables the robot to change its transformations depending on environmental conditions. When traversing on continuous and smooth steel surfaces, the robot activates the *mobile transformation* as shown in Fig. 3.2(a). The robot navigates using a path planning algorithm with the help of differential wheels and performs visual inspection of steel bridge structures. Moreover, the robot can move on an inclined steel surface by the adhesion forces supported by two magnetic arrays mounted on each robot foot. There are two working modes of the magnetic arrays: *touched* and *untouched*, indicating the distance from the magnetic arrays to the steel surface where the robot feet lies on. *Touched mode* means the distance is zero, and the *untouched* one keeps the distance around $1mm$. The *mobile transformation* requires both magnetic arrays operating in untouched mode to generate two magnetic adhesion forces, which are enough for the robot standing on the inclined surface, and in same time still let the robot can move by its wheels. The

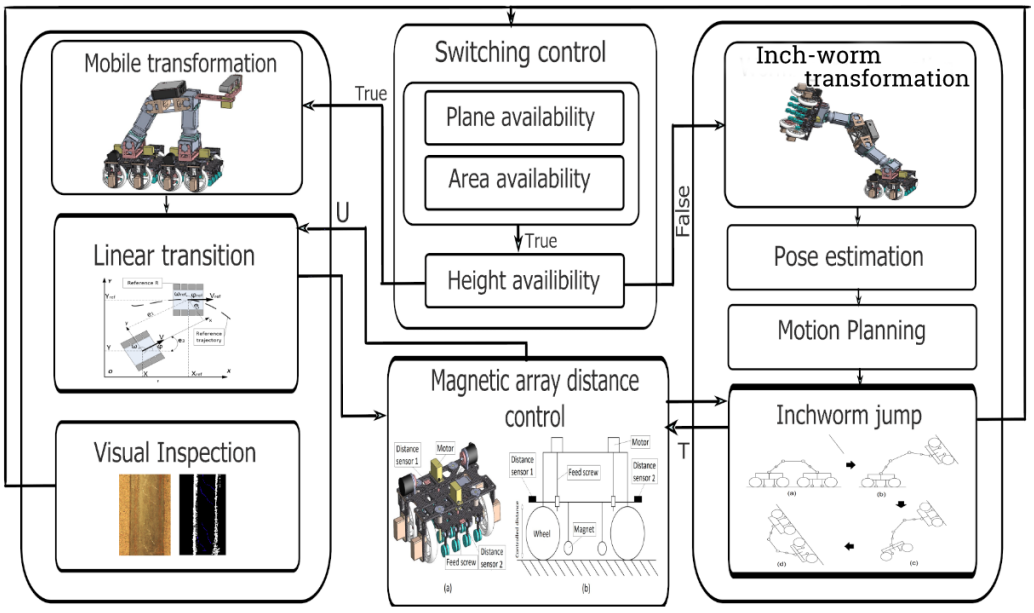


Figure 3.1: The proposed control system framework for autonomous navigation

robot switches into *inch-worm transformation* (Fig. 3.2(b)) when it detects a complex steel surface and cannot move on wheels, then activates an inch-worm jump to the next surface. As performing inch-worm jump, only one of the robots feet touches the steel surface. To create enough adhesive force for the robot standing, the magnetic array is switched to *touched mode*, which fully allows this array to adhere the steel surface. The switching control mechanism controls the movement of the robot, detects environment type and sends the appropriate command to executable nodes.

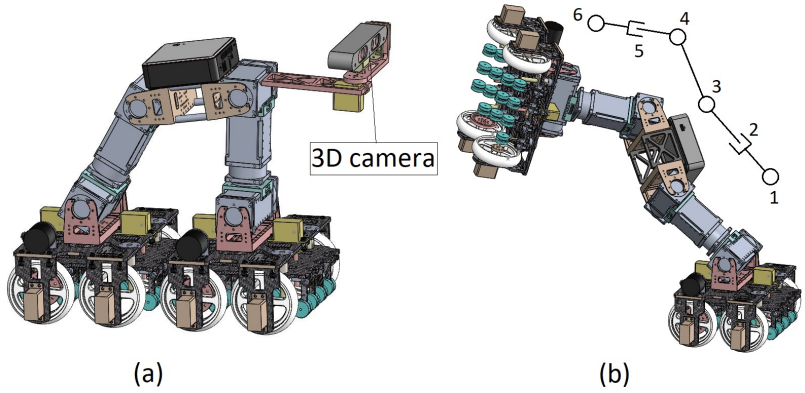


Figure 3.2: ARA robot in (a) *mobile* and (b) *inch-worm transformation*

The control architecture of ARA robot is comprised of multiple low-level and high-level control structures [20]. Several tasks are performed by the low-level control structure (Arduino). The wheel's velocity, encoder reading, and the magnetic array function are performed in this control level. The high-level control embedded in an on-board processor manages switching control function, point cloud data processing, inverse kinematics and motion planning. The arrangement of the high level and low-level controls is shown in Fig. 3.3.

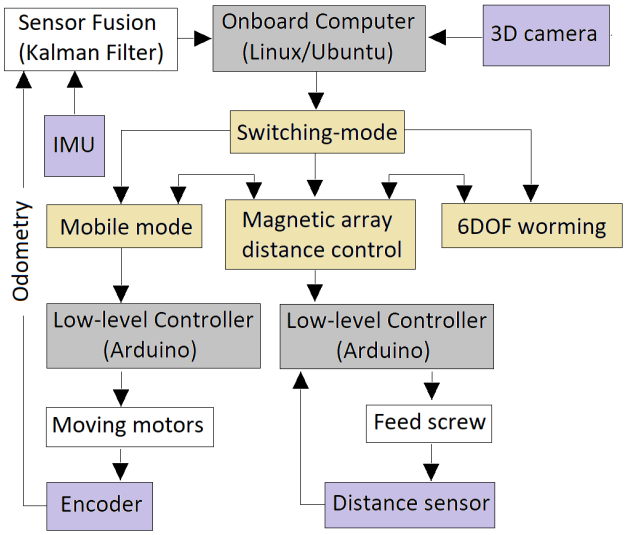


Figure 3.3: The control architecture integrated into ARA robot [1]

The ARA robot control framework consists of four modules: *switching control*, *inch-worm transformation*, *mobile transformation*, and *magnetic array control*. An overview of the overall framework is shown in Fig. 3.1. The first two modules, which are the contributions of the author's work will be presented in detail.

3.1 Switching Control

The switching control S enables the robot to autonomously configure itself into two transformations (mobile and inch-worm). The control employs switching function S,

represented in Eq.3.1. The function takes as input three Boolean parameters: plane availability S_{pa}, area availability S_{am} and height availability S_{hc}. These parameters determine if there is any still surface available, while enabling the estimation of the area of the surface and its height. A logical operation is performed on these parameters using function $f(.)$. The function's parameters are estimated from 3D point cloud data of steel surface.

$$S = f(S_{pa}, S_{am}, S_{hc}) = S_{pa} S_{am} S_{hc}. \tag{3.1}$$

The robot configures into *mobile transformation* if the function S returns a true value. The false value configures the robot into the *inch-worm transformation*.

Plane availability: The 3D PCL of a steel surface is processed using *pass-through* filtering, *downsampling*, and *plane detection* [54]. The *plane detection* applied the RANSAC method [55] extracts the planar point cloud P_{cl} from the initial point cloud. The plane availability is checked using Eq. (3.2):

$$\begin{cases} S_{pa} = False, \, if \; P_{cl} = \emptyset \\ S_{pa} = True, \, otherwise. \end{cases} \tag{3.2}$$

Moreover, two functions *get_centroid* and *get_normal_vector* provide the point cloud's centroid $C_{P_{cl}}$ and normal vector $\vec{N_{P_{cl}}}$ of the point cloud.

Area availability: The robot feet requires an area estimation of the available planar surface area P_{cl}. It is essential to ensure the availability of sufficient area for successful robot transition. This is a popular problem in legged-robot, which has been investigated carefully in [35–37]. In [35, 36], the authors proposed the convex-based algorithms, which deployed convex optimization problem to determine an obstacle-

free ellipsoid (convex one), then estimate step-able areas for a biped robot. In [37], the authors proposed an algorithm to determine the valid convex collision-free regions with geometrical constraints of obstacles. In those algorithms, a portion of the step-able area, especially as the vertex number is small, was not considered due to the convex approximation. It is a problem for our inspection robot with large feet pair due to limited step-able areas on steel bridges. Those algorithms may not be possible to find a step-able area for the ARA robot to worm in many cases. Thus, we developed two algorithms, which can process a non-convex plane boundary efficiently to estimate a step-able area for the ARA robot. *Algorithm 1* extracts a non-convex boundary from the planar point cloud, then the second one - *Algorithm 2* checks the sufficiency of the available planar surface.

Algorithm 1 Non-convex boundary point estimation from 3D point cloud data of steel bridges

procedure BOUNDARYESTIMATION(P_{cl}, α_s)
  $Planes = \{xy, yz, zx\}$
  $d_{min} = \forall_{i \in Planes}$ //Point along minimum value of plane i
  $d_{max} = \forall_{i \in Planes}$ //Point along maximum value of plane i
  Initialize $B_s = \{\}$
  for $p \in Planes$ **do**
    $i \rightarrow 1$
    while $sl_{p_i} < d_{max}$ **do**
      $sl_{p_i} = d_{min_p} + i * \alpha_s$
      PS_{p_i} = Set of points in range $sl_{p_i} \pm \alpha_s/2$
      $P_{cl_A}, P_{cl_B} = \underset{\forall\{P_i, P_j\} \in PS_{p_i}}{\text{argmax}} \{\|P_i - P_j\|\}$
      $B_s = B_s \cup \{P_{cl_A}, P_{cl_B}\}$
      $i = i + 1$
    end while
  end for
return B_s
end procedure

The boundary points estimated in *Algorithm 1* are performed by a window-based approach. The algorithm's input is the point cloud P_{cl} of the estimated planar surface and a slicing parameter α_s. At first, we calculate the two furthest points represented as

d_{min} and d_{max} in the point cloud P_{cl} along each plane, then the point cloud is divided into multiple smaller slices along the three planes. For each slice in a particular plane p, the slicing index sl_p is determined, which represents the center coordinate of the slice as shown in line 9 of algorithm 1. After that, the point sets PS_p in the range $sl_p \pm \alpha_s/2$ is extracted from P_{cl}. This sliding factor is experimentally determined based on the point cloud size. For each set of points from PS_p, two furthest points (P_{cl_A}, P_{cl_B}) are extracted. These two points are estimated as the boundary point for that particular slice and added to the boundary point sets B_s. This approach helps the algorithm works well with plane with small holes inside. A pictorial representation of the boundary estimation algorithm is shown in Fig. 3.4.

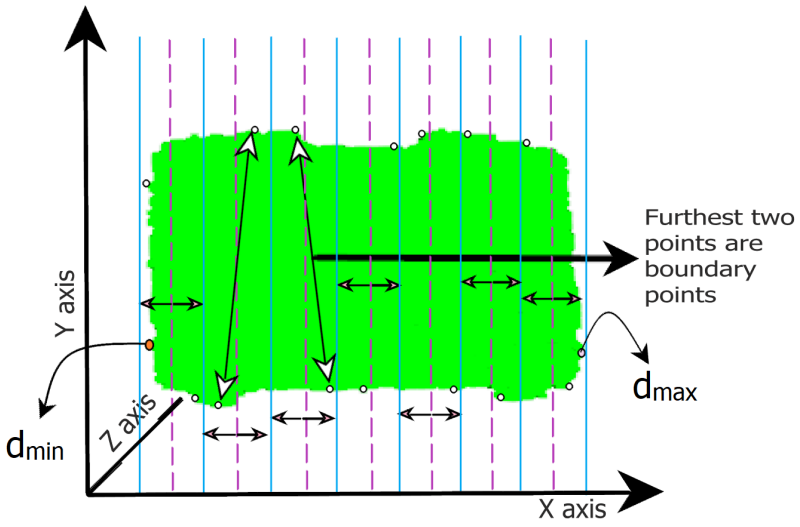


Figure 3.4: Boundary point estimation from 3D point cloud data

After determining the boundary points B_s, the area availability variable S_{am} is estimated by using *Algorithm 2*. The inputs are the boundary points B_s, point cloud centroid $C_{P_{cl}}$, normal vector of point cloud $\vec{n_{P_{cl}}}$, length l and width w of robot feet, and distance tolerance t. At first we calculate the n closest points (N_{clos}) from B_s to the point cloud centroid $C_{P_{cl}}$. For each point, N_i in the set N_{clos}, a set of computations is performed to estimate the plane corners for adherence to robot wheels. The

Algorithm 2 Area Checking and Pose Estimation

procedure AREA($B_s, C_{P_{cl}}, n\vec{_{P_{cl}}}, w, l, t, S_{am}$)
  N_{clos} = Find n closest points to $C_{P_{cl}}$ from B_s
  for $N_i \in N_{clos}$ **do**
    $R = \{\}$, //Estimated rectangle corner points
    $\vec{e_{x_i}} = N_i - C_{P_{cl}}$
    $\vec{e_{z_i}} = n\vec{_{P_{cl}}}$
    $\vec{e_{y_i}} = \vec{e_{x_i}} \times \vec{e_{y_i}}$
    $k_w = \frac{w}{|\vec{e_{x_i}}|} e_{y_i}$ and $k_l = \frac{b}{|\vec{e_{y_i}}|} e_{x_i}$,
    $\{R_1, R_2\} = \{N_i + k_w, N_i - k_w\}$
    $R = R \cup \{R_1, R_2\}$
    $R = R \cup \{R_1 + k_l, R_2 + k_l\}$
    $M = \forall_{r_i \in R} \{\frac{r_i + r_{i+1}}{2}\}$
    $R = R \cup M$
    $S_{am} = True$
    for $r_i \in R$ **do**
      Q_i = Find m closest points to r_i
      $d_{r_i} = \|d_{r_i}, C_{P_{cl}}\|$ and $d_{Q_i} = \|Q_i, C_{P_{cl}}\|$
      $S_i = (d_{r_i} < d_{Q_i}) \vee (\frac{d_{r_i} - d_{Q_i}}{d_{r_i}} < t$
      $S_{am} = S_{am} \wedge S_i$
    end for
    Pose = {Orientation, Position}
    if S_{am} == True **then**
      R_c = Centroid of R
      Orientation = $(\vec{e_{x_i}}, \vec{e_{y_i}}, \vec{e_{z_i}})$
      Position = $(x_{R_c}, y_{R_c} - l/4, z_{R_c})$
      return Pose
    end if
  end for
  return False
end procedure

coordinate frame vectors $\vec{e_{x_i}}, \vec{e_{y_i}}$ and $\vec{e_{z_i}}$ are calculated for point N_i. In the next step, the algorithm estimates the corner points of a rectangle of width w and length l, which is also robot foot's width and length, respectively. We estimate the rectangle's edges parallel along the vectors $\vec{e_{x_i}}$ and $\vec{e_{y_i}}$. Therefore, the four corners R of the rectangle are estimated using these two vectors. Additionally, we include four middle points of the estimated rectangle corners in R to alleviate point cloud collection error as well as accommodate for the non-convex shape of the steel surface. After the estimation step, we find m closest points to R from the B_s to measure if the points in R are inside the boundary. Hence, we calculate the distance from point cloud centroid $C_{P_{cl}}$ to R and Q, respectively. The algorithm considers a point, which lies inside the boundary if its tolerance is less than t and the distance to centroid should be less than its neighbors. The algorithm's performance is presented in Fig. 5.2(b)-(d). When the value of S_{am} is true for all the conditions, we consider those sets of points as rectangular points.

Height availability: The height availability S_{hc} is crucial for the switching control. Based on this parameter, the switching control activates the robot transformations. At first the point cloud's centroid $C_{P_{cl}}$ is calculated along the camera frame f_c. Then it is transformed to the robot base frame f_{rb} using Eq. 3.3.

$$P_{C_{f_{rb}}} = T_{f_{rb}f_c} P_{C_{f_c}}, \tag{3.3}$$

where ($P_{C_{f_{rb}}}$, $P_{C_{f_c}}$) are coordinates of the centroid C_c in the camera frame and the robot base frame, respectively. $T_{f_{rb}f_c}$ is the transformation matrix from the camera frame f_c to the robot base frame f_{rb}.

The plane height $z_{f_{rb}}$ coordinate is then compared to the robot base height. If they are equal, the returned result is *true*, and the robot is configured as *mobile transformation*. Otherwise, it returns *false*, and the robot go to *inch-worm transformation*.

The height availability condition is shown as in Eq. 3.4.

$$\begin{cases} S_{hc} = True, \, if \, z_{f_{rb}} = z_{robotbase} \\ S_{hc} = False, \, otherwise. \end{cases} \tag{3.4}$$

3.2 Inchworm transformation

The inch-worm transformation enables the robot to perform an inch-worm jump from one steel surface to another as shown in Fig. 3.5. At first, the permanent magnet array on the second foot of the robot is set to *touched mode*, which adheres the leg on steel surface and generates a strong adhesive force for the robot to stand and perform the worming. A controller manipulates the joints to move the first robot leg towards the target plane as shown in Fig. 3.5(b). As the first leg touches the target surface, the first and second permanent magnetic arrays are switched to *touched mode* and *untouched mode*, respectively. After that, the second leg detaches from the starting surface as in Fig. 3.5(c). Finally, in Fig. 3.5(d) the second leg is adhered the target surface.

Converting from *mobile* (Fig. 3.2(a)) to *inch-worm transformation* is challenging for the motion planner to create a trajectory. To have a better performance, a convenient robot pose P_{conv} is proposed where the robot should move to firstly as starting of *inch-worm transformation*. From there, the motion planner will generate a trajectory for the first leg to move the destination. The worming is completed by moving the second leg to the target surface and reform *mobile configuration*. To have the target pose, the robot needs to determine the target plane and its pose, which are the outputs of *Algorithm 2*.

The flexibility of the robot in worming is made of the six DoF arm. The revolute joints of the arm in Fig. 3.2(b) can rotate along three different axes separately. For

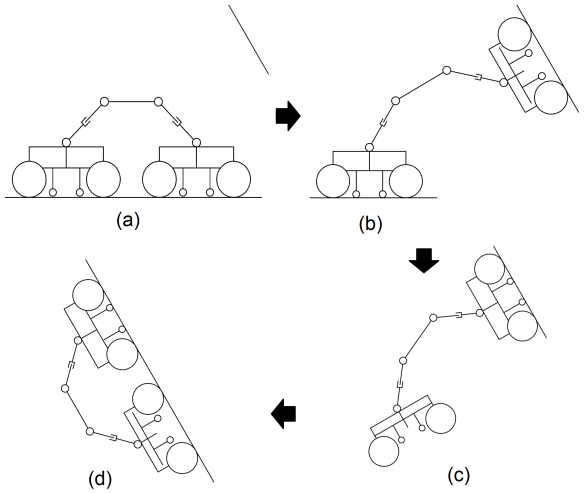


Figure 3.5: Inch-worm jump from one steel surface to another

example, the joint 2 and joint 5 in Fig. 3.2(b) are configured to rotate around *y-axis*. The rest of the joints are positioned to rotate around *z-axis*. This configuration was selected for maintaining symmetry so that the manipulator can move efficiently in both worming and mobile mode. Our previous research states in detail elaboration of the robotic arm in [1].

3.3 Summary

In this chapter, the control framework and the detail of two modules: *Switching Control* and *Inchworm transformation* were described. The Switching control helps the robot change the operation types, which make the robot move in two types of steel bar surfaces: smooth and continuous, and inclining. The Inchworm transformation module determines the target surface, its pose and generates a motion planning for the ARA robot movement.

Chapter 4

Navigation Framework

As the ARA robot works on *mobile transformation*, it traverses all steel bars of a steel bridge to detect the failure. To do the task in shortest time, it is needed to solve an optimization problem to determine the shortest path, which goes through all the available steel bars at least once [56]. The task is sent to the motion planning module, and the inputs are point cloud (PCL) data, target position, and robot location from SLAM. The PCL data of the steel bridge is filtered and then projected into a 2D plane, which is the *xy* plane of the robot coordinate frame [20]. The structure segmentation algorithm - *Algorithm 3* separates the processed data into the clusters, then those boundaries are estimated by *Algorithm 2* - Non-Convex Boundary Estimation (NCBE) [20]. Graph construction algorithm - *Algorithm 4* processes the boundaries data to build a graph that represents the steel bridge structure, then *Algorithm 5* - VOCPP algorithm solves the optimization problem to find the shortest path. The VOCPP algorithm can solve any graph with any starting and ending points. A single-query path planning such as RRT receives the cluster boundaries and shortest path as inputs and builds a traverse path for the ARA robot to go along the steel bars. In this

path planning, we propose *Algorithm 6* called *Point Inside Boundary Check - PIBC* to determine efficiently the collision and availability of the robot configuration by the boundaries. The output of the motion planner is sent to the ARA robot controller, which regulates the robot at the lower level to perform the motion. The framework is shown in Fig. 4.1.

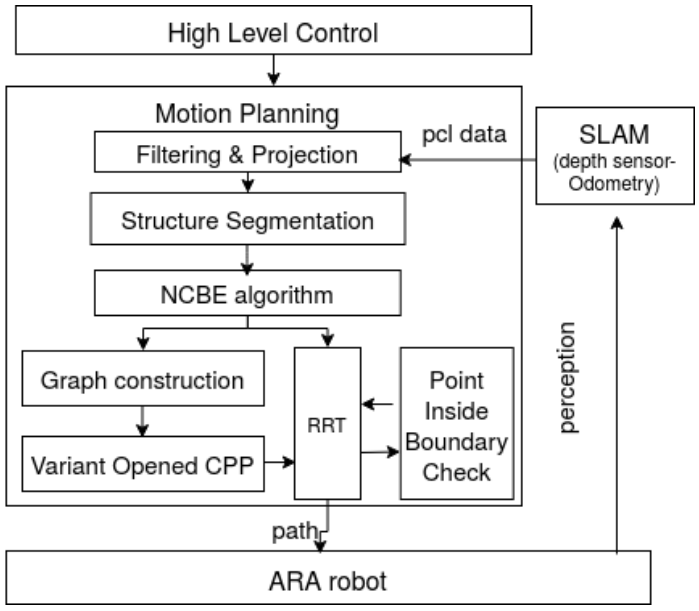


Figure 4.1: The proposed navigation framework on mobile mode

4.1 Steel Bridge Structure Segmentation

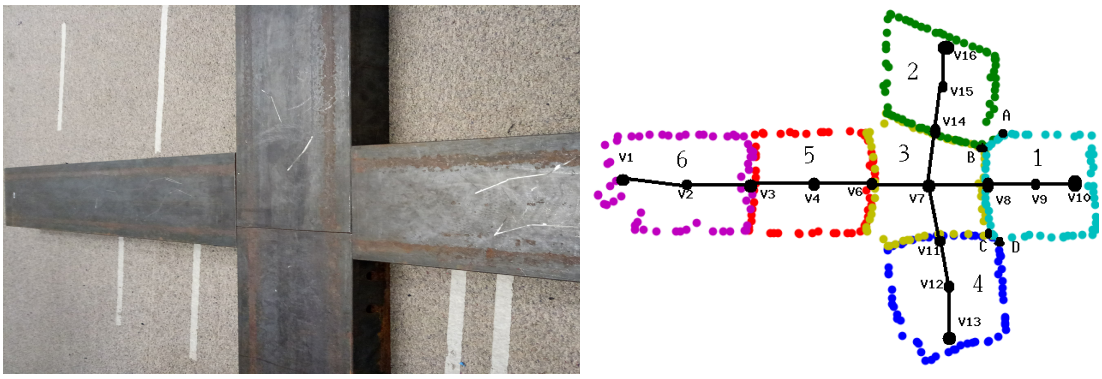
Motion planning for a robot traversing on a steel bridge is a challenging task because of the limitation of working space (steel bar's dimensions) and the steel bridge structure's diversity. Serious damage occurs if the robot moves out of the steel bar edges. Therefore, steel bridge perception is critical to build a motion planner. The bridge structure can be perceived by a convolutional neutral network (CNN) [57], however, it requires a huge data set and expensive tasks such as labeling. Moreover, with the variety of the structures, in the best of the author's knowledge, there is no CNN

working on steel bridge structure detection.

To detect the steel bridge structure, we propose a method based on the bridge's geometric features that they typically consists of two components: (1) steel bars and (2) cross areas. As the method segments the structure into these two components, the bridge structure is represented by a graph whose edges and vertices are the steel bars and cross areas, respectively. The steel bars are irregular in shape with one dimension being much longer than other, and their two ends connect two cross areas. The cross areas are regular in shape and its dimensions are similar, and there are at least two steel bars connected with it. The difference in the geometric features of the steel bars and cross areas are used to classify them.

From the feature analysis, an algorithm is developed based on EM-GMM classification [38, 40, 41]. The EM-GMM classification is able to separate the structure into multiple clusters with irregular shapes. However, it requires a specific cluster number as input, and if there are several types of distribution in the data, EM-GMM does not works well. To deal with unknown cluster numbers, the algorithm works on a set of cluster numbers and determines which is the most suitable number n_o for cluster segmentation. The number n_o is then inputted into the EM-GMM algorithm [38, 40] that generates a set of clusters.

To find the best cluster number for the steel bridge segmentation, a new concept - neighbor cluster - is introduced. *Two clusters are considered as neighbors if they share a border with the length at least* l_b. From that, in Fig. 4.2b, cluster 1 and 3 are considered neighbors because the border BC is longer than the threshold l_b. Although there are some contacting points, there can be no neighbor-relationship between clusters 1 and 2 because the border AB is shorter than l_b. The same applies to cluster 1 and 4 since $CD \leq l_b$. From this idea, if the cluster numbers n_o are optimal, the cross area should be the one with most neighbor clusters n_m as shown in



(a) (b)

Figure 4.2: (a) A *Cross-* shape steel bar (camera view on the right) and (b) its segmentation

Fig. 4.2b (cluster 3). Therefore, the first idea is that the cluster number n_o is optimal if it makes the cross area cluster having n_m with highest value.

The most neighbor cluster numbers n_m works well for *K-*, *T-*, *Cross-* shape. However, for the structures such as *I-* and *L-* shapes, the highest number of neighbor is not enough to segment correctly. For *L*-shape, as the cluster number is more than three, there is several clusters with the same neighbor cluster number, and it is not possible to find the cross area cluster. For *I*-shape, there is two cross-area cluster in the structure. Thus, it is needed another feature to handle these shapes, and the difference between the most neighbor number n_m and the second most neighbor number n_s with the same n_c is considered. Combining the two features, the ratio r is defined in Eq. (4.1), and the highest r will be selected.

$$r = \frac{n_m}{n_m + n_s} + \frac{n_m}{n_c}. \tag{4.1}$$

In Eq. (4.1), the first term tends to keep n_c small, and the second term tends to make n_m large. n_c is the number of clusters.

The procedure detail is presented by the psudo-code in *Algorithm 3*. The inputs

Algorithm 3 Steel Bridge Structure Segmentation

1: **procedure** PCL SEGMENTATION($P_{cl}, N_{cmin}, N_{cmax}$)
2: Initialize $n_{index}, r_n = \emptyset$
3: **for** $n_c \in \{N_{cmin}, N_{cmax}\}$ **do**
4: $S_c = EM\text{-}GMM\ Algorithm(P_{cl}, n_c)$
5: **for** $j \in \{0, n_c\}$ **do**
6: $b[j] = NCBE(S_c[j], sliding\_factor)$
7: **end for**
8: Determine neighbor clusters for $S_c[j]$
9: Get the most and second most neighbor numbers n_m, n_s
10: Calculate r from Eq. 4.1, and add to set r_n
11: Add the n_c in set n_{index}
12: **end for**
13: Select the highest r in r_n, and get n_o from n_{index}
14: S_b = EM-GMM Algorithm(P_{cl}, n_o)
15: **return** S_b
16: **end procedure**

are the PCL data, and a range of cluster number from N_{cmin} to N_{cmax}. As n_c runs in the range of [N_{cmin} to N_{cmax}], EM-GMM algorithm segments the PCL data into a set of clusters S_c. From line 5 to 7, NCBE algorithm determines the set of boundaries b corresponding to S_c. The neighbor number for each cluster is determined in line 8, and from that the most and second most cluster numbers are specified. From n_c, n_m, and n_s, the ratio r is calculated, then the r and n_c are putted into the sets r_n and n_{index}, respectively. Step 13 selects the highest r, which is inputted into EM-GMM algorithm to get the most appropriate segmented clusters S_b.

4.2 Graph Construction and VOCPP

From an inspection requirement aspect, the ARA robot should monitor all available steel bars in the bridge structure to check for any failure. As the cross areas and steel bars are considered as nodes and edges, respectively, the inspection requirement is able to be solved by *inspection route problem* or Chinese Postman Problem (CPP) [53]. Therefore, a graph representing the structure is constructed, then the optimization

problem is solved to find the shortest inspection route for the robot.

Algorithm 4 Graph Construction

1: **procedure** GRAPH CONSTRUCTION(data cluster set S_{bo}, threshold d_{min})
2: Initialize $E = \{\}, V = \{\}$, current vertex v_c
3: Calculate the center point set S_c of boundary set S_b
4: Determine neighbor matrix M_n
5: Determine borders set S_{bd} among the neighbors from S_c, M_n
6: Determine the middle point set S_{md} of S_{bd}
7: Add all vertices S_c, S_{md} into V: $V = S_c \cup S_{md}$
8: Build edges e by connecting vertices in V if its corresponding cluster are neighbors (from M_n
9: Estimate line set S_l to fit the cluster set S_c by PCA
10: Determine intersection of feature lines & their boundaries: $I_l = S_l \cap S_b$
11: **for** i $\in I_l$ **do**
12: **for** j $\in V$ **do**
13: **if** distance from $I_l[i]$ to $V[j]$ larger than d_{min} **then**
14: Add $I_l[i]$ into V
15: Edge e = from $I_l[i]$ to its center vertex
16: Add e into E
17: **end if**
18: **end for**
19: **end for**
20: **return** $G = (V, E)$
21: **end procedure**

Algorithm 4 builds the graph from the cluster boundaries set S_{bo} (from Algorithm 1) and a distance threshold d_{min}. Firstly, the center points S_c of clusters, neighbor matrix M_n, and border middle points S_{md} are determined from step 3 to 6. All the points in S_c and S_{md} are added into the vertices set V. The edge set is built by M_n and V in step 8. After that, a set of feature's lines that are fit to the boundaries are calculated by PCA method [58]. Step 10 determines the intersection set I_l of the feature's lines and their cluster boundary S_b. From step 11 to 19, there are dual loops to check whether the distance from a point in I_l to a vertex in V is longer than the threshold d_{min}. If yes, then that point and an edge, which connects it to its center point, are added into the vertex set V and the edge set E, respectively. The algorithm outputs $G = (V, E)$, which is sent to *Algorithm* 5.

Algorithm 5 Variant Open CPP

procedure MINIMAL PATH($G = (\mathcal{V}, \mathcal{E}), v_s, v_t$)
  Initialize $\mathcal{G}_m, S_{mov}, S_{ov}$
  Find all odd vertices and add to S_{ov}.
  if $S_{ov} = \emptyset$ **then**
    Find shortest edge set $\mathcal{E}_e$ connecting v_s and v_t
    Assign $\mathcal{G}_m = (\mathcal{V}, \mathcal{E}_{\Updownarrow} = \mathcal{E} \cup \mathcal{E}_e)$, go to 24.
  end if
  if $(v_s, v_t) \subset S_{ov}$ **then**
    $S_{mov} = S_{ov} \backslash \{v_s, v_t\}$, $G_m = G$, go to 23.
  end if
  if $v_s \notin S_{ov}$ and $v_t \in S_{ov}$ **then**
    Select a vertex $v_c \in S_{ov}$, which creates the shortest path $\mathcal{E}_c$ connecting v_s to it
    $S_{mov} = S_{ov} \backslash \{v_t, v_c\}$, $\mathcal{G}_m = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_c)$, go to 23
  end if
  if $v_s \in S_{ov}$ and $v_t \notin S_{ov}$ **then**
    Select a vertex $v_c \in S_{ov}$, which creates the shortest path $\mathcal{E}_c =$ connecting v_t to v_c
    $S_{mov} = S_{ov} \backslash \{v_s, v_c\}$, $\mathcal{G}_m = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}_c)$, go to 23
  end if
  if $v_s \notin S_{ov}$ and $v_t \notin S_{ov}$ **then**
    Select two vertices $v_{c1}, v_{c2} \in S_{ov}$, which creates two paths $\mathcal{E}_{d1}, \mathcal{E}_{d2}$ that sum of them is shortest
    $S_{mov} = S_{ov} \backslash \{v_{c1}, v_{c2}\}$, $G_m = \{\mathcal{V}, \mathcal{E} \cup \mathcal{E}_{d1} \cup \mathcal{E}_{d2}\}$, go to 23.
  end if
  Using $\mathcal{G}_m$, find a set of edges $\mathcal{E}_a$ whose sum is shortest to convert all vertices in S_{mov} to even vertices.
  Find the Eulerian path P_{robot} from $\mathcal{G}_n = (\mathcal{V}, \mathcal{E}_m \cup \mathcal{E}_a)$
return P_{robot}
end procedure

After constructing the graph, the next step is to find the shortest path to inspect all the available steel bars. As the robot inspects the bridge parts step by step, the starting and ending point should be different. Moreover, the steel bridge structures are diverse, thus the input graph could consist of Euler circuit, Euler path, or none. Algorithm 5 - Variant Open CPP (VOCPP) is proposed to solve the optimization problem that is based on *Euler Theorem* [59] for Eulerian trail. The algorithm converts any input graph into an *open CPP* graph by adding the shortest path into the input graph and makes a new one. The added paths are selected by using Dijkstra's algorithm [60]. The algorithm's output is the shortest path, which visits each edge

at least once and starts and ends at the predefined positions.

The *pseudo-code* is shown in Algorithm 5. The inputs are the graph $G = \{V, E\}$, starting and target vertex v_s, v_t. Step 4 checks whether an Eulerian circuit exists in the graph. If yes, a path generated by Dijkstra's algorithm with two vertices v_s, v_t is added into the graph. After that, the algorithm will jump to step 24. If the odd vertex set S_{ov} are not empty, step 8 checks whether v_s, v_t belong to the set. If yes, both are popped out, and a set of shortest paths is added to convert all the remained odd nodes in S_{ov} to even. Steps 11, 15, and 19 check whether the vertex v_s or v_t stays in the S_{ov}, or if both are out of the set. If any input vertex is odd, it will be popped out. Vertices in S_{ov} that possess the shortest paths to the even input vertices are connected by a shortest path to convert the even input vertex into odd. After that, the selected vertex is also popped out of S_{ov}. All then go to step 23 to convert all the odd vertices in the remaining S_{ov} into even vertices by Dijkstra's algorithm. Step 24 will find the shortest path that traverses all edges of the graph by Fleury's algorithm [59].

4.3 Point Inside Boundary Check - PIBC

After receiving the shortest path from the VOCPP algorithm, a motion planning is deployed to generate a path to control the robot to traverse all the defined edges. It plans the motion path for one edge each time.

In our scenario, the robot moves on a set of steel bars with assumption that there is no obstacle on it. The free configuration, however, is limited by the space of the steel bar surface. The serious damage can occur if the robot moves out of the steel bar edges. It is a hard constraint to construct a free configuration C_{free} for the robot. To utilize all available configurations, the point cloud boundary is applied to build

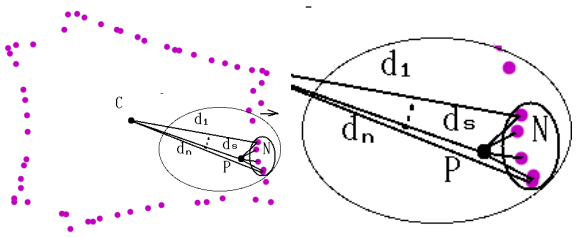


Figure 4.3: Point inside the boundary check

the obstacle-free configuration. After receiving the boundary set S_{bo} from NCBE algorithm, a robot configuration c_i belongs to the free configuration C_{free} if all of its projected points S_{pp} lie inside one of the steel bar boundaries $b_i \in S_{bo}$.

Algorithm 6 Point Inside Boundary Check

```
1: procedure SAMPLE CHECK(S_bo, c, P)
2:     Calculate the cluster center point set c_p of each boundary in S_bo
3:     Project from configuration c to the robot position set PS
4:     Initialize boolean set bs[len(PS)] = Fail
5:     for i ∈ (0, len(PS)) do
6:         Calculate distances d_s from PS[i] to c_p
7:         Select n center points, which are closest to PS[i]
8:         for p ∈ (0, n) do
9:             Find m_p closest points to PS[i] in each S_bo[p]
10:            for j ∈ m_p do
11:                Calculate distance d_{m_pj} from m_p[j] to c_p[p]
12:                if d_s < d_{n_pj} then
13:                    bs[i] = True
14:                end if
15:            end for
16:        end for
17:    end for
18:    for i ∈ (0, len(PS)) do
19:        if bs[i] = Fail then return Fail
20:        end if
21:    end for
22: return True
23: end procedure
```

To determine whether a point lies inside a steel bar boundary, a geometric technique named *center-closest points* is applied as shown in Fig. 4.3. A sample point P is considered inside the boundary if its distance d_s to the center point C is shorter

than the distances d_n of its closest neighbors to the center. The closest neighbor set N is determined by finding the minimal Euclidean distance set.

The implementation of the technique is presented in *Algorithm 6*. The inputs are the boundary set S_{bo}, robot configurations c, and robot parameter set P. From step 5 to 17, the loop runs through all the robot configurations. For each configuration, it finds a set of clusters in which it possible belongs to. For each cluster, the technique *center-closest points* is deployed to specify whether it lies inside that boundary. If the configuration stays inside only one of the cluster boundary, its corresponding boolean variable $bs[]$ will set *True*. Step 18 uses a *for* loop to check whether there is any robot configuration not belonging to any cluster boundary. The algorithm returns *True* if any configuration lies inside a boundary.

4.4 Summary

In this chapter, the navigation framework for the ARA robot on mobile transformation is presented. It consists of four algorithms: *Steel Bridge Structure Segmentation*, *Graph Construction*, *VOCPP*, and *PIBC*. Steel Bridge Structure Segmentation algorithm determines the most appropriate cluster number to segment the steel structure, which is essential to build a graph by Graph Construction algorithm. *VOCPP* algorithm uses the outputted graph and generates the shortest path, which traverses all the steel bars at least one, with any starting and ending points. The last algorithm *PIBC* is an integrated portion of motion planner, which determines whether a robot configuration point lies inside the steel bars.

Chapter 5

Experiment and Results

In this chapter, the experiments and results are presented and discussed. The experiments were performed indoor with several samples of steel bridge structures. Due to the lack of equipment and limited indoor space, there are two sets of steel bridge structures were built to test the ARA robot on two transformations: *Inchworm* and *Mobile*, which are discussed in section 5.1. The results are discussed in section 5.2.

5.1 Experiment Setup

The experiments were implemented on the ARA robot version 1.0, which was derived from [1] with an additional camera module. An RGB-D camera (ASUS Xtion Pro Live) is attached to the robot for point cloud collection and visual inspection. The camera calibration's parameters were integrated from the method implemented by [54]. The robot was localized by *aruco marker*, which was placed on the robot standing surface. We performed a geometric calculation to locate the position between the aruco marker and the robot base. Additionally, an Intel NUC 5 - Core i5 vPro

was incorporated for employing the robotic operating system (ROS) as well as the inspection framework.

The first experiment would test the control framework, which can switch the robot transformations, and control the robot to perform the inchworm jump, and run on smooth surfaces. Two steel slabs located perpendicularly from each other were set up for this task (Fig. 5.4). The steel slabs are highly corroded to replicate steel defects. The second experiment was on the robot navigation as running on smooth surfaces. Several steel bridge structures such as *K*-, *L*-, *I*-, *T*-, and *Cross*- shapes were assembled on the ground (Fig.5.5 a-e) to simulate the typical steel bar structures. The following section describe the experiments and their results elaborately.

5.2 Results

The results of two experiments - control and navigation frameworks were presented and discussed. They are two critical portions, which constituted the autonomous system for the ARA robot.

5.2.1 Switching control

At starting, the PCL data of a steel bridge structure sample was collected from the robot camera. An example of the initial PCL was shown in Fig.5.1(a). After performing some pre-processing operations such as *pass-through filtering* and *downsampling*, the data was sent to *plane detection* to extract the planar surface. The processed PCL was shown in Fig.5.1(b). The coordinate frame was also shown in the figure with x-axis in red, y-axis in green, and z-axis in blue.

After obtaining the planar surfaces, the surface boundary points and *Area availability* checking were performed by *Algorithm 1* and *Algorithm 2* on two different

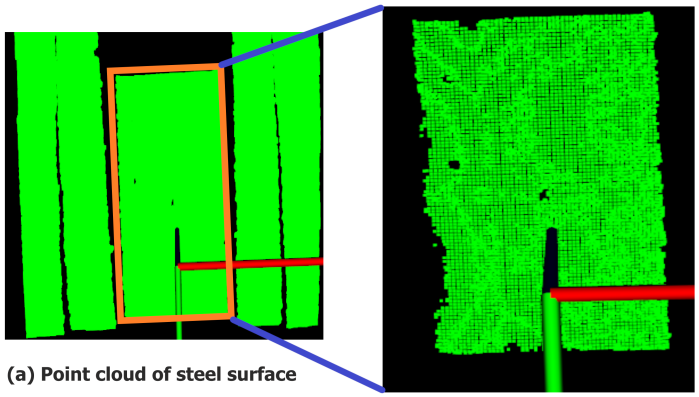
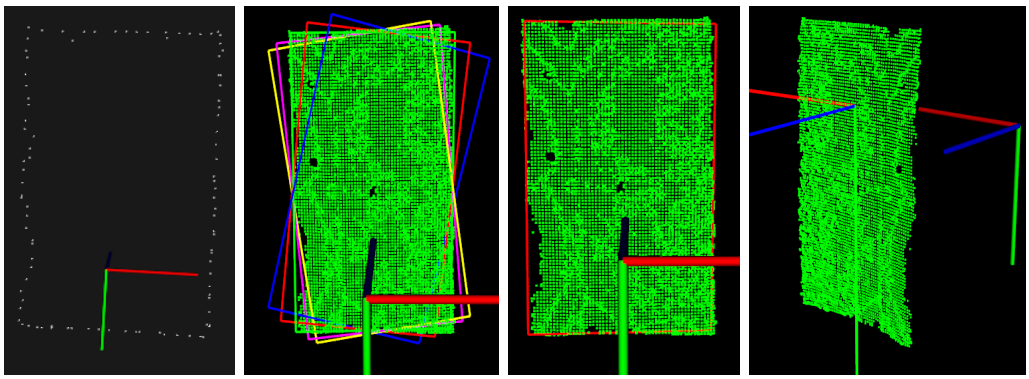


Figure 5.1: Planar surface extraction from 3D point cloud of steel surface surfaces, one containing sufficient area for movement and the other without. Using *Algorithm 1*, two boundary points of the two different point cloud as shown in Fig.5.2(a) and Fig.5.2(c).



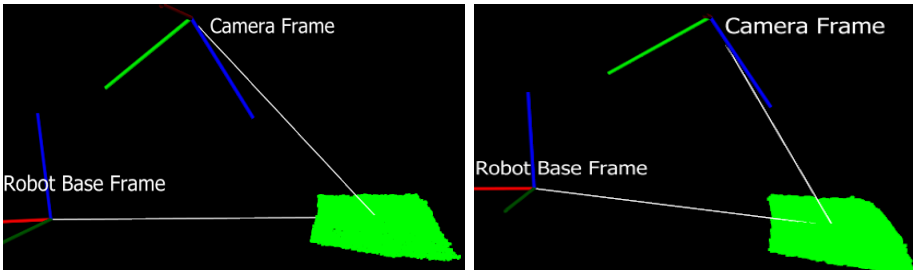
(a) (b) (c) (d)

Figure 5.2: (a) Boundary set, (b) Area rectangle set, (c) The selected area rectangle, and (d) Pose estimation

The area availability check from *Algorithm 2* was employed using the boundary point estimated. The algorithm parameters were as following : $n = 5, m = 3$ and $t = 0.02$. Five rectangles were estimated for the robot feet with this algorithm as shown in Fig. 5.2(b) in red, yellow, blue, green, and purple color. In Fig. 5.2(b), several corners of all the red, yellow, purple, and blue rectangles were outside of the point cloud area. It represented that these rectangles area were not sufficient

enough for an inch-worm jump. Only the green rectangle was inside the point cloud, satisfying area requirement. The selected rectangle is shown in Fig. 5.2(c) (in red color). Since there is enough area for an inch-worm jump, the variable S_{am} is set to true by the algorithm. After that, the planar surface pose was estimated as shown in Fig. 5.2(d) with three orientations shown in red, green, and blue color, respectively on the point cloud surface.

The surface pose was then transformed into the robot base frame. If the pose's height (corresponding to z- axis in the robot base frame) was equal to robot base height, the value of S_{hc} was set to *True*, and the robot configured itself as *mobile transformation*. Fig. 5.3(b) represented another scenario as the point cloud is from a surface, which was $d = 7cm$ lower than the robot base. In this case, the heights were different, then the returned value of variable S_{hc} was *false*, and robot performs *inch-worm transformation* in the next step.



(a) (b)

Figure 5.3: Surface Height Check (a) Same Height & (b) Different Height

5.2.2 Inchworm Transformation

KDL Inverse Kinematics and *RRTConnect* motion planner in *MoveIt* package were selected to implement the task, which calculated the robot inverse kinematics and generated a trajectory for an inch-worm jump from point P_{conv} to the target plane. To do that, a primitive robot model was built in *urdf* format, with the exact dimensions,

joint types and limits to *ARA* robot. The generated trajectory - a ROS topic - was a set of robot joint angles, which the robot joints followed to reach the target pose.

The inch-worm performance of the robot was shown in Fig. 5.4. In the beginning, the robot activated and lowed down the magnetic array to touch the steel surface; then it transformed from the mobile configuration to the convenient pose P_{conv} by following a predefined trajectory as shown in Fig. 5.4(a) and Fig. 5.4(b). As reaching point P_{conv}, the robot started following the *RRTConnect* trajectory. As the first foot reached the target surface, the robot switched both magnetic arrays working modes, the one on the first foot was changed to *touched mode*, and the other was set to *untouched mode* as shown in Fig. 5.4(c)-(d). Next, the second robot foot transformed into the target plane as shown in Fig. 5.4(e)-(f). The whole robot operation was filmed, and the video-clip was uploaded at `https://youtu.be/SHk5IIOBRdA`, which was sped up three times than the experimental operation.

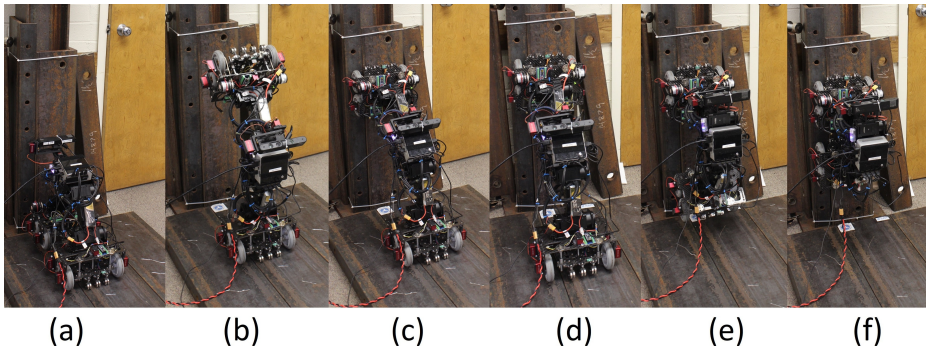


Figure 5.4: *inch-worm transformation*: a) magnetic array of second foot touched the base surface, b) first foot moved to convenient point, c) first foot reached target pose and touched the second surface, d) magnetic array of second foot was released, e) and f) second foot moved to target pose

5.2.3 Navigation Framework

In this experiment, the four algorithms in section 4 were tested on steel bridge structures such as *L*-, *Cross*-, *T*-, *K*, and *I*- shapes. Due on the lab condition, we combined

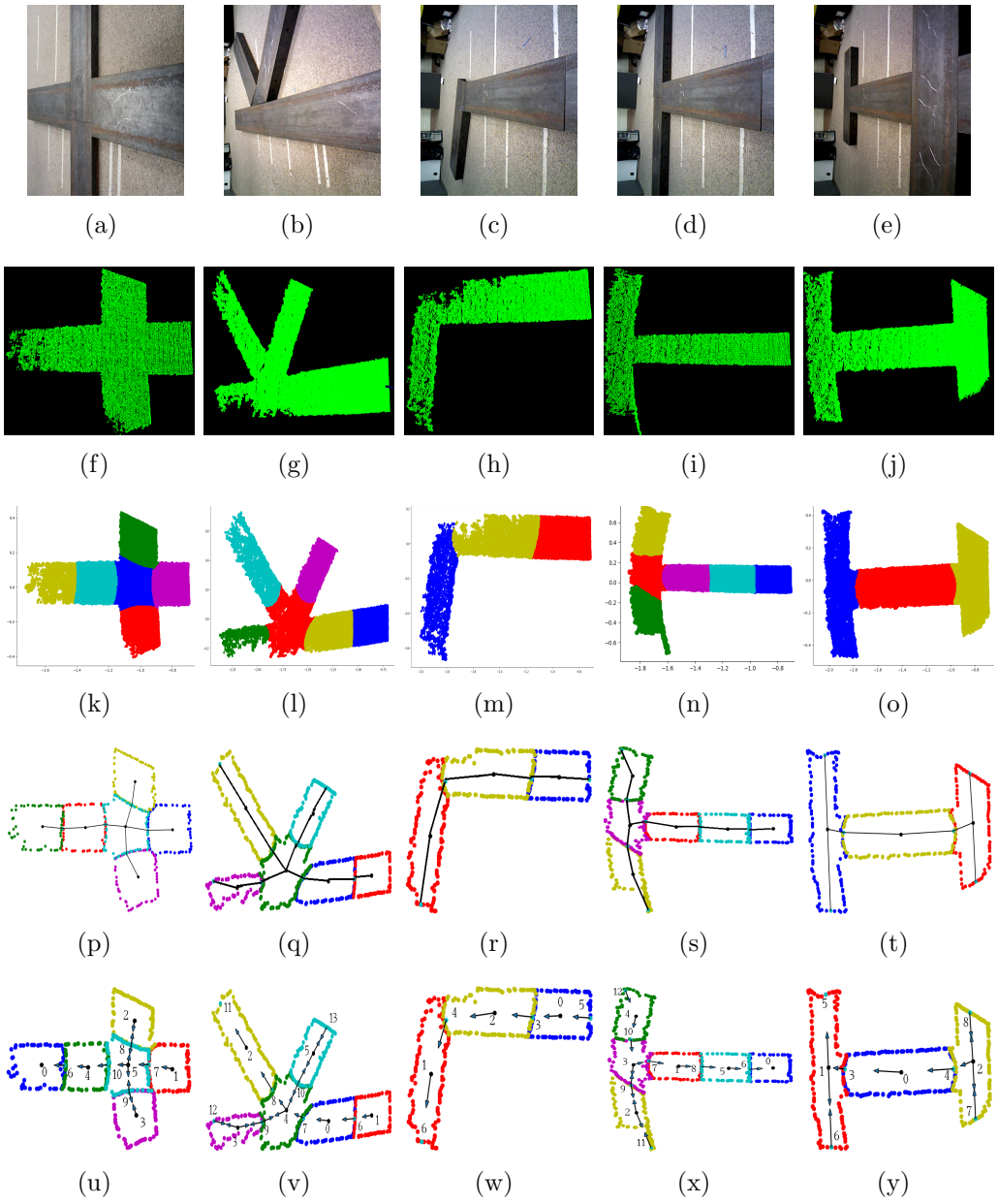


Figure 5.5: The images of input structures (a-e), the corresponding point clouds (f-j), the segmentation (k-o), boundary estimation and graph construction (p-t), and the shortest path (u-y)

several steel bars on the ground to make the shape types mentioned as shown in the first row of Fig. 5.5. To present the experiment results succinctly, the structure images and PCL data were rotated 90 degree with the viewpoint from the right to left.

Structure Segmentation and Boundary Estimation

The result of the segmentation algorithm was shown in Fig. 5.5. The images in the first and second rows were the RGB and point cloud data of *Cross-*, *K-*, *L-*, *T-*, and *I-* shapes after filtering and projected into the robot coordinate frames. The view-point of camera was from right to left, and the farther to the camera, the more degrading of sensory data quality. The efficiency of *Algorithm 3* was shown in the third row of Fig. 5.5. The algorithm ran well and was able to segment properly the cross areas and the steel bar parts, except the *I-* shape. The reason was that there were two cross areas in *I*-shape structure, which made *algorithm 3* confuse. This problem will be improved in the future research. By a robust graph construction algorithm, however, the graph of *I-* shape could be still built and helped generate the path for the robot. After segmenting, the clusters were sent to the *NCBE algorithm* [20], which worked efficiently to give back the boundaries for the cluster.

Graph Construction and VOCPP

Graph Construction (*Algorithm 4*) and VOCPP (*Algorithm 5*) processed the boundary data from the *NCBE algorithm*, and outputted the result on the fourth and fifth rows, respectively. In the fourth row, the graph was built based on the center point, edge points of the boundary and the border points between the neighbor clusters. The graphs covered the steel bars' length for most structures, except the *Cross-* structure (Fig. 5.5p). It was because of the distance from the center points to the corresponding edge points were too close, the depth sensor could cover all the distance. Therefore, it was no need an edge to connect the two points. In Fig. 5.5v, the edge (12-3) did not go along with the steel bar but crossing on one edge. It occurred because the data quality was not good, and influencing the line fitting algorithm. The same reason happened to the *T-* structure in Fig. 5.5s.

The figures in the last row of Fig. 5.5 show the shortest path generated by the

VOCPP algorithm. Due to the probabilistic feature of EM-GMM algorithm, the cluster indices changed each time running. Starting at a random vertex v_s, the robot followed the arrow lines to the next vertex, and comes back if there was a dead end. The route ended at the predefined ending vertex v_t, and the generated paths were optimal with shortest length. Again, due to the point cloud data quality, in Fig. 5.5v,x, the edges *(3,12)*, *(2,11)* were not possible for robot to traverse. To prevent the robot go on the edges, the motion planner was needed to handle this case.

Point Inside Boundary Check - PIBC and Path Planning

Using PIBC algorithm with RRT motion planner, a motion path was generated as shown in Fig. 5.6 in L-shape structure. The algorithm was significantly affected by the data quality, however, it still worked well to determine whether a robot configuration belong to the free space. To reduce the processing time for the robot, RRT motion planning [61, 62] was deployed in the robot.

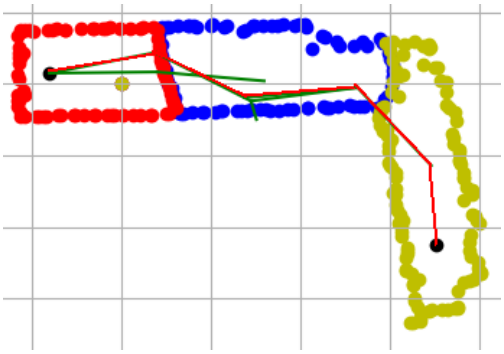


Figure 5.6: Result of Algorithm 6

Chapter 6

Conclusion and Future Work

In this chapter, the research results are summarized and highlighted the major contribution. Moreover, some problem arose in the experiments showed the needed improvement for the robot to run autonomously on real steel bridge structures.

6.1 Conclusion

In this thesis, control and navigation frameworks were proposed for ARA robot. In control framework, a switching control mechanism for autonomous navigation of bridge-inspection robots was developed. The unique feature of switching in two modes enhanced the flexibility of navigation and inspection. The most significant part of this framework are two algorithms: *Algorithm 1 - Non-convex Boundary Estimation (NCBE)* and *Algorithm 2 - Area Availability* to detect and determine area, plane and height availability.

The navigation framework was developed for ARA robot to run on mobile transformation. In this transformation, the robot needed to cross and inspect all the steel

bars. The major portions of this framework were four algorithms, which could process the depth data, then outputted a traverse path for the robot. *Algorithm 3 - Structure Segmentation* segmented the steel bar structures into two sets: steel bars and cross areas. Based on the segmentation result, the graph construction - *Algorithm 4* built a graph that represents the bridge structure. The graph is inputted to *Algorithm 5 - VOCPP* to generate the shortest path for the robot to move and inspect all the available steel bars. *Algorithm 6* helped RRT path planning generate a trajectory, which the robot controller regulated the robot to follow.

The two proposed frameworks are crucial portions to build an autonomous framework, which makes ARA robot implementing the tasks on the real steel bridge structure. To get a full autonomous framework for ARA robot, there are still several problems, which needed to be solved and improved, as described in the following section.

6.2 Future Work

Due to the complexity of Inchworm manipulator-like structure, the motion planner *RRTConnect* in *Inchworm transformation* was not robust in calculating the inverse kinematics and generating the trajectory for the robot, and needed to redo sometimes. This limits the jumping operation of ARA robot in complex steel bridge structure. Further investigation of deployment in actual steel bridges, building a new motion planner for this robot, and optimization of inch-worm transformation is necessary as the next phase of the research.

In the navigation framework, the efficiency of the algorithms needs to extend to process more bridge structures. The stability of the segmentation algorithm, which depends on the probability method, is not in well-operating and sometimes outputs

inappropriate segmentation. To solve that, the formula 4.1 could need to be refined, or a CNN needs to be developed to provide better results for this algorithm.

Moreover, the integration of multiple portions into a single framework to handle the real-world environment was the most challenging part of this research. It needs a cooperation of a team with interdisciplinary knowledge and skills to let ARA robot run on a real steel bridge.

Bibliography

[1] S. T. Nguyen, A. Q. Pham, C. Motley, and H. M. La, "A practical climbing robot for steel bridge inspection," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9322–9328.

[2] P. Prasanna, K. J. Dana, N. Gucunski, B. B. Basily, H. M. La, R. S. Lim, and H. Parvardeh, "Automated crack detection on concrete bridges," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 591–599, 2016.

[3] U. H. Billah, H. M. La, and A. Tavakkoli, "Deep learning-based feature silencing for accurate concrete crack detection," *Sensors*, vol. 20, no. 16, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/16/4403

[4] H. Ahmed, H. M. La, and N. Gucunski, "Review of non-destructive civil infrastructure evaluation for bridges: State-of-the-art robotic platforms, sensors and algorithms," *Sensors*, vol. 20, no. 14, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/14/3954

[5] H. Ahmed, H. M. La, and K. Tran, "Rebar detection and localization for bridge deck inspection and evaluation using deep residual networks,"

Automation in Construction, vol. 120, p. 103393, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0926580520309730

[6] R. S. Lim, H. M. La, and W. Sheng, "A robotic crack inspection and mapping system for bridge deck maintenance," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 2, pp. 367–378, 2014.

[7] Ronny Salim Lim, H. M. La, Zeyong Shan, and Weihua Sheng, "Developing a crack inspection robot for bridge maintenance," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 6288–6293.

[8] H. M. La, N. Gucunski, Seong-Hoon Kee, J. Yi, T. Senlet, and Luan Nguyen, "Autonomous robotic system for bridge deck data collection and analysis," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1950–1955.

[9] H. M. La, R. S. Lim, B. Basily, N. Gucunski, J. Yi, A. Maher, F. A. Romero, and H. Parvardeh, "Autonomous robotic system for high-efficiency non-destructive bridge deck inspection and evaluation," in *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, 2013, pp. 1053–1058.

[10] H. M. La, N. Gucunski, K. Dana, and S.-H. Kee, "Development of an autonomous bridge deck inspection robotic system," *Journal of Field Robotics*, vol. 34, no. 8, pp. 1489–1504, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21725

[11] N. Gucunski, S.-H. Kee, H. La, B. Basily, A. Maher, and H. Ghasemi, *Implementation of a Fully Autonomous Platform for Assessment of Concrete Bridge Decks RABIT*, pp. 367–378. [Online]. Available: https://ascelibrary.org/doi/abs/10.1061/9780784479117.032

[12] S. Gibb, H. M. La, T. Le, L. Nguyen, R. Schmid, and H. Pham, "Nondestructive evaluation sensor fusion with autonomous robotic system for civil infrastructure inspection," *Journal of Field Robotics*, vol. 35, no. 6, pp. 988–1004, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21791

[13] H. M. La, R. S. Lim, B. B. Basily, N. Gucunski, J. Yi, A. Maher, F. A. Romero, and H. Parvardeh, "Mechatronic systems design for an autonomous robotic system for high-efficiency bridge deck inspection and evaluation," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 6, pp. 1655–1664, 2013.

[14] T. Le, S. Gibb, N. Pham, H. M. La, L. Falk, and T. Berendsen, "Autonomous robotic system using non-destructive evaluation methods for bridge deck inspection," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3672–3677.

[15] S. Gibb, T. Le, H. M. La, R. Schmid, and T. Berendsen, "A multi-functional inspection robot for civil infrastructure evaluation and maintenance," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2672–2677.

[16] T. Seo and M. Sitti, "Tank-like module-based climbing robot using passive compliant joints," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 1, pp. 397–408, 2012.

[17] H. Wang and A. Yamamoto, "Analyses and solutions for the buckling of thin and flexible electrostatic inchworm climbing robots," *IEEE Transactions on Robotics*, vol. 33, no. 4, pp. 889–900, Aug 2017.

[18] S. T. Nguyen and H. M. La, "Development of a steel bridge climbing robot," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1912–1917.

[19] H. La, "Steel climbing robot with magnetic wheels," Mar. 26 2020, US Patent App. 16/464,874.

[20] H.-D. Bui, S. Nguyen, U. Billah, C. Le, A. Tavakkoli, and H. M. La, "Control framework for a hybrid-steel bridge inspection robot," in *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, Nevada, USA, October 25 – 29*, 2020.

[21] S. T. Nguyen and H. M. La, "Development of a steel bridge climbing robot," in *Intelligent Robots and Systems, 2019. IROS 2019. IEEE/RSJ Intern. Conf. on*, Nov 2019.

[22] A. Q. Pham, H. M. La, K. T. La, and M. T. Nguyen, "A magnetic wheeled robot for steel bridge inspection," in *Advances in Engineering Research and Application*, K.-U. Sattler, D. C. Nguyen, N. P. Vu, T. L. Banh, and H. Puta, Eds. Cham: Springer International Publishing, 2020, pp. 11–17.

[23] S. T. Nguyen and H. M. La, "Roller chain-like robot for steel bridge inspection," in *The 9th International Conference on Structural Health Monitoring of Intelligent Infrastructure (SHMII-9)*, 2019, pp. 890–895.

[24] T. Seo and M. Sitti, "Tank-like module-based climbing robot using passive compliant joints," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 1, pp. 397–408, Feb 2013.

[25] H. M. La, T. H. Dinh, N. H. Pham, Q. P. Ha, and A. Q. Pham, "Automated robotic monitoring and inspection of steel structures and bridges," *Robotica*, vol. 37, no. 5, pp. 947 – 967, May 2019.

[26] N. H. Pham and H. M. La, "Design and implementation of an autonomous robot for steel bridge inspection," in *54th Allerton Conf. on Comm., Con., and Comp.*, Sept 2016, pp. 556–562.

[27] S. Kamdar, "Design and manufacturing of a mecanum sheel for the magnetic climbing robot," *Master Thesis, Embry-Riddle Aeronautical University*, May 2015.

[28] N. H. Pham, H. M. La, Q. P. Ha, S. N. Dang, A. H. Vo, and Q. H. Dinh, "Visual and 3d mapping for steel bridge inspection using a climbing robot," in *The 33rd Intern. Symposium on Automation and Robotics in Construction and Mining (ISARC)*, July 2016, pp. 1–8.

[29] "Magghd," https://eddyfi.com/en/product/magghd/.

[30] T. Bandyopadhyay, R. Steindl, F. Talbot, N. Kottege, R. Dungavell, B. Wood, J. Barker, K. Hoehn, and A. Elfes, "Magneto: A versatile multi-limbed inspection robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 2253–2260.

[31] "Versatrax 100^{TM}," http://inuktun.com/en/products/.

[32] K. D. Kotay and D. L. Rus, "Navigating 3d steel web structures with an inchworm robot," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*, vol. 1. IEEE, 1996, pp. 368–375.

[33] Y. Xie, W. Wang, J. Guo, and K.-M. Lee, "Edge detection using structured laser pattern and vision for mobile robot navigation," in *2011 IEEE/ASME*

International Conference on Advanced Intelligent Mechatronics (AIM). IEEE, 2011, pp. 910–915.

[34] D. Pagano and D. Liu, "An approach for real-time motion planning of an inchworm robot in complex steel bridge environments," *Robotica*, vol. 35, no. 6, pp. 1280–1309, 2017.

[35] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 109–124.

[36] S. Jatsun, S. Savin, and A. Yatsun, "Footstep planner algorithm for a lower limb exoskeleton climbing stairs," in *International Conference on Interactive Collaborative Robotics*. Springer, 2017, pp. 75–82.

[37] A.-C. Hildebrandt, R. Wittmann, F. Sygulla, D. Wahrmann, D. Rixen, and T. Buschmann, "Versatile and robust bipedal walking in unknown environments: real-time collision avoidance and disturbance rejection," *Autonomous Robots*, vol. 43, no. 8, pp. 1957–1976, 2019.

[38] D. A. Reynolds, "Gaussian mixture models." *Encyclopedia of biometrics*, vol. 741, 2009.

[39] T. Pfeifer and P. Protzel, "Expectation-maximization for adaptive mixture models in graph optimization," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3151–3157.

[40] T. M. Nguyen and Q. J. Wu, "Fast and robust spatially constrained gaussian mixture model for image segmentation," *IEEE transactions on circuits and systems for video technology*, vol. 23, no. 4, pp. 621–635, 2012.

[41] K. Blekas, A. Likas, N. P. Galatsanos, and I. E. Lagaris, "A spatially constrained mixture model for image segmentation," *IEEE transactions on Neural Networks*, vol. 16, no. 2, pp. 494–498, 2005.

[42] H. Goforth, X. Hu, M. Happold, and S. Lucey, "Joint pose and shape estimation of vehicles from lidar data," *arXiv preprint arXiv:2009.03964*, 2020.

[43] G. Z. Gandler, C. H. Ek, M. Björkman, R. Stolkin, and Y. Bekiroglu, "Object shape estimation and modeling, based on sparse gaussian process implicit surfaces, combining visual data and tactile exploration," *Robotics and Autonomous Systems*, vol. 126, p. 103433, 2020.

[44] S. Kraemer, M. E. Bouzouraa, and C. Stiller, "Simultaneous tracking and shape estimation using a multi-layer laserscanner," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 1–7.

[45] H. Thimbleby, "The directed chinese postman problem," *Software: Practice and Experience*, vol. 33, no. 11, pp. 1081–1096, 2003.

[46] H. A. Eiselt, M. Gendreau, and G. Laporte, "Arc routing problems, part i: The chinese postman problem," *Operations Research*, vol. 43, no. 2, pp. 231–242, 1995.

[47] D. Ahr and G. Reinelt, "A tabu search algorithm for the min–max k-chinese postman problem," *Computers & operations research*, vol. 33, no. 12, pp. 3403–3422, 2006.

[48] G. J. McLachlan and D. Peel, *Finite mixture models*. John Wiley & Sons, 2004.

[49] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[50] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[51] D. Connell and H. M. La, "Dynamic path planning and replanning for mobile robots using rrt," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 1429–1434.

[52] D. Connell and H. M. La, "Extended rapidly exploring random tree-based dynamic path planning and replanning for mobile robots," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, p. 1729881418773874, 2018. [Online]. Available: https://doi.org/10.1177/1729881418773874

[53] J. Edmonds and E. L. Johnson, "Matching, euler tours and the chinese postman," *Mathematical programming*, vol. 5, no. 1, pp. 88–124, 1973.

[54] H.-D. Bui, H. Nguyen, H. M. La, and S. Li, "A deep learning-based autonomous robot manipulator for sorting application," in *2020 IEEE International Conference on Robotics Computing*. IEEE, 2020, p. accepted.

[55] T. Strutz, "Data fitting and uncertainty," *A practical introduction to weighted least squares and beyond. Vieweg+ Teubner*, 2010.

[56] H.-D. Bui and H. M. La, "Navigation framework for a hybrid steel bridge inspection robot," *arXiv preprint arXiv:2101.02282*.

[57] Y. Narazaki, V. Hoskere, T. A. Hoang, and B. F. Spencer Jr, "Automated vision-based bridge component extraction using multiscale convolutional neural networks," *arXiv preprint arXiv:1805.06042*, 2018.

[58] Wikipedia contributors, "Principal component analysis — Wikipedia, the free encyclopedia," 2020, [Online; accessed 30-October-2020]. [Online].

Available: https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=984408123

[59] ——, "Eulerian path — Wikipedia, the free encyclopedia," 2020, [Online; accessed 28-October-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Eulerian_path&oldid=980960712

[60] ——, "Dijkstra's algorithm — Wikipedia, the free encyclopedia," 2020, [Online; accessed 28-October-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&oldid=982111493

[61] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[62] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "Python-robotics: a python code collection of robotics algorithms," *arXiv preprint arXiv:1808.10703*, 2018.