Modeling and Roadmap Generation for Truss Inspection by Small UAS

Arun Das[1] and Craig Woolsey[2]

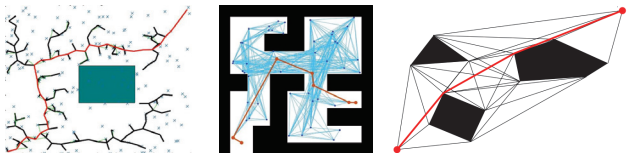
***Abstract*— Small unmanned aircraft systems (UAS) can increase efficiency and reduce risk to humans during the inspection of truss-supported structures such as steel bridges. This paper describes a method to mathematically represent a truss and subsequently to plan an efficient, collision-free path from a given starting position to a given goal. The algorithm generates a deterministic roadmap which connects nodes that are generated near the joints of the structure. This roadmap is then searched for the path using a lazy A* algorithm. Simulation results show the functionality of the algorithm. Comparison with a probabilistic roadmap method indicates the proposed algorithm's efficiency.**

I. INTRODUCTION

With continuous improvements in performance, payload capacity, and reliability, multicopters have become helpful tools for aerial mapping, event monitoring, and infrastructure inspection. Concerning inspection, multicopters have been used or proposed to image railways, pipelines and powerlines, tunnels, buildings and a variety of other infrastructural assets. Here we focus on the inspection of bridges, which are an especially important subset of the public infrastructure. Many existing bridges are supported by trusses, typically built of steel because of its strength and resilience. In the United States, bridges must be inspected every two years and the large number of welded and bolted connections between beams, girders, and other elements make this task time-consuming and costly, because of both direct labor costs and indirect costs due to traffic disruptions. Moreover, bridge inspection is inherently dangerous, providing further motivation for robotic inspection.

Various concepts have been proposed for robotic inspection of steel bridges. These include walking [6], climbing, and driving robots [7], [9] that use magnets to adhere to the structure. These systems can have difficulty, however, negotiating the corners, gusset plates, and other structural elements that are peculiar to a truss. We consider the use of a small unmanned aircraft system (UAS) to inspect truss-supported structures, such as steel bridges, by flying within the structure and collecting images of user-specified elements. In this paper, we develop an algorithm to plan a collision-free path through a truss structure from a given start point to a desired end point. The algorithm generates a roadmap which includes the start and end point as well as a collection of "navigation points" which form the basis for path planning. Although the path planner incorporates constraints and uncertainties associated with operating a multicopter near a truss structure, the aircraft dynamics are not explicitly considered.

A great variety of path planning algorithms exists [10]. Here, we briefly mention some popular approaches. Rapidly-exploring random trees (RRTs) are often used to quickly generate feasible paths, but RRTs do not produce optimal solutions. Probabilistic roadmap (PRM) approaches are resolution optimal, but a roadmap with a large number of nodes generally results in large computation times. Both approaches – the RRT and the PRM – have difficulty finding paths through narrow passages, which is problematic for operation in the confined space within a truss. By maximizing the distance to obstacles, planning methods based on Voronoi graphs emphasize path safety over path length. Potential field methods are another alternative, but they are computationally expensive. For two-dimensional environments, visibility graphs efficiently produce shortest paths by limiting the number of nodes in the graph. Visibility graphs are also effective at dealing with narrow passages. However, visibility graphs cannot be easily extended to three dimensions.



(a) RRT (b) PRM (c) Visibility Graph

Fig. 1: Path planning methods in 2-dimensional environments with polygonal obstacles. Resulting paths in red.

The proposed algorithm is based on a new method of representing the structure of steel bridges. For this representation all parts of the bridge are enclosed by cuboids and a small set of parameters is provided as input for the algorithm. These input parameters are then used by the algorithm to construct a representation of the structure. Then a deterministic roadmap is generated by computing Navigation Points (NPs) around the joints of the structure. The setting of NPs is similar to visibility graph approaches. The NPs can be used as waypoints along the path to find a feasible path from start to goal. The roadmap is searched for the path with a 'lazy' version of the A* graph search algorithm.

In Section II, the representation of steel bridges is described and the computation of the structure is shown. Afterwards, Section III describes the path planning algorithm. Simulation results are provided in Section IV, followed by a conclusion in Section V.

[1]Arun Das is with the Faculty of Electrical Engineering, Technical University Munich, 80333 Munich, Germany a.das@tum.de

[2]Craig Woolsey is a Professor in Virginia Tech's Crofton Department of Aerospace and Ocean Engineering, Blacksburg, VA 24061, USA cwoolsey@vt.edu

II. REPRESENTATION OF TRUSSES

In this section, a method is introduced to mathematically represent a truss structure. The method enables one to efficiently compute occupied and free regions within the environment from a small set of parameters that contain all the relevant information about the structure.

As a motivating example for truss inspection, a steel bridge is constructed from many different parts including beams, braces, girders, stiffeners, pillars, cables, sway bars, etc. For the purpose of modeling for path planning, every such element is enclosed by a cuboid. Multiple elements that are densely packed within a volume may also be grouped together and enclosed in a single cuboid. It is not necessary to distinguish the individual properties of different components; we treat the cuboids as standardized objects that represent obstructions within the work space. We refer to these cuboids as *beams* and we call the junctions where beams connect *joints*. While a given cuboid will often enclose a true beam within the structure, it might also enclose some other element such as a cable or walkway.

A. Coordinate Frames

To model the truss structure, we define two types of reference frames: a single, arbitrary *world frame*, labeled with the subscript *0*, and a *beam frame*, denoted by the subscript *b*, for each beam in the structure. The beam frames are needed to parameterize the truss structure in order to compute paths. The origin $\mathbf{O}_\text{b}$ is located at one of a given beam's joints, denoted the "start joint", and the z-axis of the frame points to the other joint, the "end joint". This definition is illustrated in Fig. (1), where $\mathbf{p}_\text{start}$ and $\mathbf{p}_\text{end}$ define the position of the beam's start and end joint, respectively. The x-axis is defined by taking the cross product of the unit vectors defining the z-axis of the world frame and the z-axis of the beam frame, according to (2). If this cross product is zero, then $\mathbf{x}_\text{b} = \mathbf{y}_0$. In order to obtain a right-handed coordinate system, $\mathbf{y}_\text{b}$ is given as in (3).

$$\hat{\mathbf{z}}_\text{b} = (\mathbf{p}_\text{end} - \mathbf{p}_\text{start})/\|\mathbf{p}_\text{end} - \mathbf{p}_\text{start}\| \quad (1)$$

$$\hat{\mathbf{x}}_\text{b} = \begin{cases} (\mathbf{z}_0 \times \mathbf{z}_\text{b})/\|\mathbf{z}_0 \times \mathbf{z}_\text{b}\| & \text{if } \mathbf{z}_0 \times \mathbf{z}_\text{b} \neq \mathbf{0} \\ \mathbf{y}_0 & \text{otherwise} \end{cases} \quad (2)$$

$$\hat{\mathbf{y}}_\text{b} = (\mathbf{z}_\text{b} \times \mathbf{x}_\text{b})/\|\mathbf{z}_\text{b} \times \mathbf{x}_\text{b}\| \quad (3)$$

B. Input Parameters

The positions of joints and parameters of beams must be provided as input to the structure-modeling algorithm. The joints are defined by their Cartesian coordinates given in the world frame. Each beam is a simple cuboid defined by six parameters: the start joint n_start and end joint n_end, the cross-sectional dimensions x_size and y_size, and an offset given by x_offset and y_offset.

The cross-sectional dimensions x_size and y_size must be greater than zero. The offset allows for cases where the centerline of a beam does not pass through the start and end joint.

Beyond the geometric parameters introduced above, we define a binary classification of joints and beams as either *active* or *inactive*, depending on whether paths around the component are allowed. Inactive joints and beams may correspond, for example, to structural elements at the boundaries of the inspection area for which no inspection data are required. Inactive components are not considered by the algorithm when generating the roadmap, reducing its size and saving computation time.

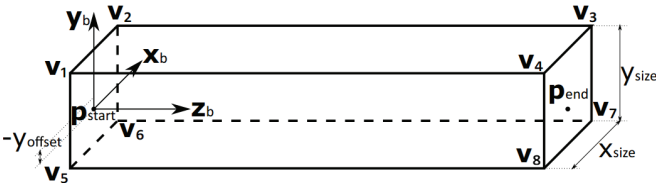


Fig. 2: A beam with corresponding parameters and vertices.

C. Defining the Structure from Input Parameters

With the given input parameters, the algorithm is computes eight vertices $\mathbf{v}_\text{i}$ for $i \in \{1, 2, \ldots, 8\}$ for each beam with respect to the beam frame. Fig. 2 shows a beam with its beam frame, parameters, and vertices. The coordinates of the first vertex with respect to the beam frame are

$$\mathbf{v}_1 = \begin{pmatrix} -\frac{x_\text{size}}{2} + x_\text{offset} \\ \frac{y_\text{size}}{2} + y_\text{offset} \\ 0 \end{pmatrix} \quad (4)$$

The remaining three vertices adjacent to the beam's starting joint are computed analogously to the first. The vertices adjacent to the beam's end point (on the right side in Fig. 2) are computed similarly, but have the z-value $\|\mathbf{p}_\text{end} - \mathbf{p}_\text{start}\|$.

After computing the vertices with respect to the beam frame, each vertex $\mathbf{v}_i$ is transformed into the world frame by applying the homogeneous transformation

$${}^0\mathbf{T}_b(\mathbf{v}_i) = \begin{bmatrix} \hat{\mathbf{x}}_\text{b} & \hat{\mathbf{y}}_\text{b} & \hat{\mathbf{z}}_\text{b} \end{bmatrix} \mathbf{v}_i + \mathbf{p}_\text{start}$$

Given the computed vertices, an object for each beam is created in the world frame and the structure is obtained by aggregating all beams.

D. Inflating the Structure

A UAS is exposed to a wide variety of external effects while flying. These range from uncertainties in position estimation to wind impacts and turbulence near the bridge structure. To limit the risk of collisions the UAS has to maintain a certain distance d_buffer to the structure. The chosen distance d_buffer depends on wind strength, controller capabilities and the disposition of taking risk, just to name some criteria. Further, the algorithm assumes the UAS to be a point. To account for the physical dimensions of the UAS, the point must maintain an additional distance from the structure, at least half the diameter d_UAS of the smallest sphere that can enclose the vehicle.

To guarantee that the planned path always keeps the required distance to the structure, the algorithm *inflates* the structure [2] in every dimension by the inflation size $d_\text{inflation}$.

This procedure is also known as *dilation* [4] or *growing the obstacles* [8]. The inflation size $d_{\text{inflation}}$ is

$$d_{\text{inflation}} = d_{\text{buffer}} + \frac{d_{\text{UAS}}}{2} \tag{5}$$

Inflation is usually done by building the Minkowski sum [3] of the obstacles with a sphere with radius $d_{\text{inflation}}$. In this approach, inflation transforms a cuboid obstacle into a sphere, which is not especially useful in the given application. In our algorithm, to maintain a cuboid representation of obstacles, we adopt an alternative inflation approach which is also more computationally efficient. We increase the dimensions of every beam:

$$\begin{aligned} x_{\text{size,new}} &= x_{\text{size}} + 2d_{\text{inflation}} \\ y_{\text{size,new}} &= y_{\text{size}} + 2d_{\text{inflation}} \end{aligned}$$

The vertices of the inflated beams are then computed and objects representing these structural members are created, as described in Section II-C.

Beams are not inflated in the z-direction, as this would require a computationally more intensive computation of NP and would not improve the quality of paths. Thus, one special case of concern is that of a cantilever beam. Because such beams are uncommon in the truss structures of interest, one can manually extend these elements when defining the structure in order to prevent the resulting path from coming too close to the cantilever beam.

III. PATH PLANNING ALGORITHM

Once the structure is inflated, the configuration space is divided into free and occupied regions. This is the basis for our path planning algorithm, which is described in this section. First a roadmap with deterministically computed NPs is built. Afterwards, this roadmap is searched for a path in an iterative process of lazy path planning [1]. In each iteration an A* algorithm finds a path. This path is then checked for collisions, and the roadmap is updated. This is repeated until a collision free path is found.

A. Computing a Roadmap

NPs are computed relative to the joints connecting beams; only active joints and beams are considered in this step. The essential idea is to identify the "corner" between intersecting beams and to set NPs at both sides of this corner as shown in Fig. 3a.

Note that if a beam is rotated about its centerline, NPs do not need to lie on the surface of the inflated structure; see Fig. 3b. This is in contrast to visibility graphs, for which waypoints always lie on the surface of an obstacle. Another special case, that of abutted beams with differing cross-sections, as shown in Fig. 3c, is discussed shortly. The mathematical computation of NPs is described in the Appendix.

After computing all NPs for each pair of active beams at each active joint, a collision check is performed to determine if any NP lies inside the inflated structure. Points inside the structure are omitted from the roadmap. Fig. 4 illustrates the resulting NPs for an example joint with four beams.

In addition to the NPs, the start and goal points for which a connecting path must be found are included as nodes in the roadmap.

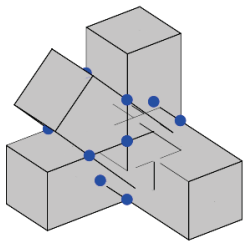
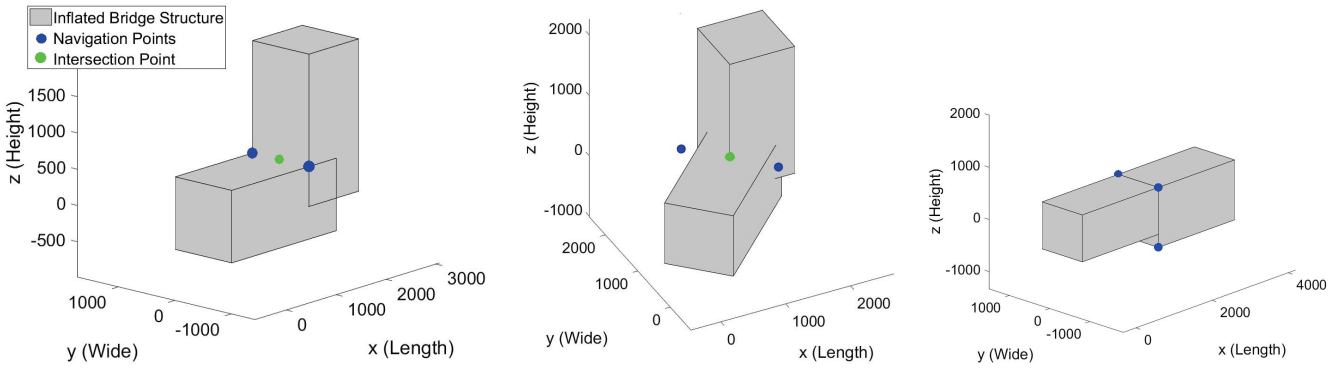


Fig. 4: NPs for a joint with multiple beams.

B. Searching for a Path

The final task for the algorithm is to find a path from the start to the goal. Our objective is to minimize distance traveled rather than to reduce risk, energy, or time, although one could easily accommodate alternative optimization criteria.



(a) Right-angled beams. (b) Rotated beams. (c) Beams in the same direction.

Fig. 3: Navigation Point (NP) examples for different combinations of beams.

The initial roadmap is a complete graph. In searching this roadmap for a minimum-distance path, we adopt a *lazy* approach [1]: no collision checks are performed for the initial roadmap. Rather, every edge connecting two nodes of the roadmap is assumed to be feasible. The weights of the edges are initialized with the Euclidean distances between the nodes.

Based on the initial roadmap an iterative process of finding the path begins. In every iteration the roadmap is searched for the shortest path from start to goal by applying an A* algorithm [5]. Edges which are part of the path are then checked for collisions and deleted from the roadmap if they result in a collision with the structure. The next iteration uses the updated roadmap to obtain a path. The process ends when a collision-free path is found.

IV. SIMULATIONS

This section describes simulation results for a half-span of the George P. Coleman Memorial Bridge located in Yorktown, VA in the United States; see Fig. 5. All simulations were conducted on a computer with an Intel Core i7-6500U 2,5GHz processor running Windows 10. The algorithm was implemented in Matlab.

A. Model of the George P. Coleman Memorial Bridge

The Coleman Bridge exhibits a repeating pattern of structural assemblies so, for path planning purposes, it is not necessary to model the entire structure. The red box in Fig. 5 indicates the assembly that was modeled, as shown in Fig. 6. In practical applications, the model would be generated from detailed construction plans. For the example considered here, a construction plan was unavailable, so the model was generated using publicly available imagery along with a few measurements obtained directly during field experiments.

The model comprises 33 active joints, 18 inactive joints, 104 active beams, and 4 inactive beams. The joints at the x-values -11000 and 66000 are inactive joints, so no NPs are set there. Further, the pillar, the deck, and the catwalks are modeled using inactive joints and inactive beams. Despite this rather simplistic modeling of the assembly, the representation enables accurate planning of collision-free paths through the structure.

Fig. 5: George P. Coleman Memorial Bridge.

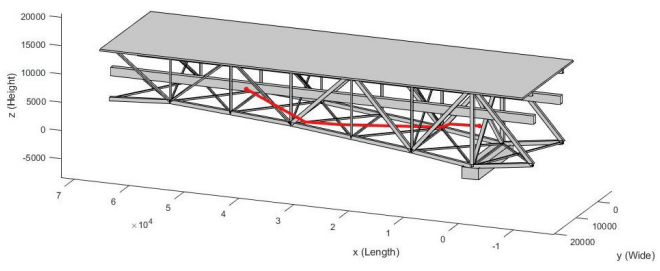


Fig. 6: Coleman Bridge model with a computed path in red.

B. Results

We ran simulations with inflation sizes ranging from 2 mm to 2000 mm. For larger inflation sizes, the cross-bracing elements contained within the span create a virtual solid wall in the model, so that path planning becomes infeasible. The results are averaged over 50 random runs for each inflation size and are summarized in Table I. In the table, the proposed algorithm is referred to as the deterministic roadmap (DRM) algorithm. For the structure of the Coleman Bridge, 694 NPs are automatically computed. An example of a path that was planned between two manually selected points can be seen in Fig. 6. The path uses 4 NPs as waypoints to connect the start and the goal. The computation took 158 s.

The distribution of computation time among the steps of the algorithm is shown in Fig. 7. This distribution is averaged among all the simulations that were conducted using the DRM algorithm. Setting up the structure and computing the NPs accoutns for a small fraction of the total computation time. Most of the time (77.7%) is consumed by the A* algorithm searching the roadmap. Checking for collisions is the second costliest activity, accounting for 18.9% of the total runtime. Abandoning the lazy approach in order to reduce the number of A* runs to one would significantly increase the number of collision checks. For simpler paths, in particular, this basic A* approach is far more time consuming than the lazy approach we propose.

The algorithm runtime is plotted versus inflation size as the blue trace in Fig. 8. With increasing inflation size, the path computation generally takes longer. Predictions of the runtime cannot be made, however, as these depend strongly on the position of the start and the goal.

To evaluate the performance of the proposed algorithm we compare with a PRM approach, a commonly used path planning method. In our implementation of the PRM approach, NPs are randomly defined throughout the structure. A total of 3000 NPs were defined, in order to balance the

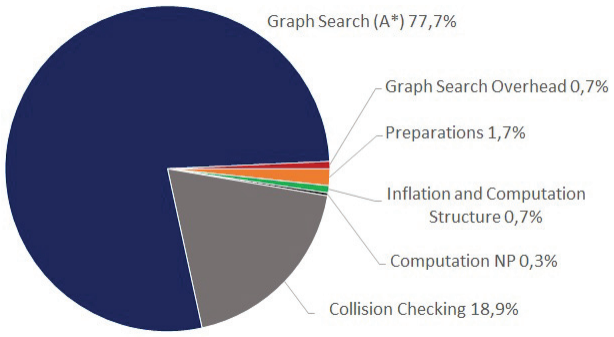


Fig. 7: Time distribution among the steps of the algorithm, averaged above inflation sizes between 2 mm and 2000 mm.

TABLE I: Results of the path planning algorithm for different inflation sizes $d_{\text{inflation}}$

Inflation Size $d_{\text{inflation}}$ [mm]	2	250	500	750	1000	1500	2000
Computation Time DRM [s]	1.9	2.6	19.7	24.7	80.1	87.3	122.3
Path Length DRM [m]	22	23	26	25	28	25	23
Computation Time PRM [s]	8.1	9.8	109.2	634.9	1120.1	-	-
Path Length PRM [m]	22	23	26	25	29	-	-

desire for low computation times with the need to negotiate narrow passages. The results from the PRM approach are also listed in Table I. The PRM algorithm's runtime is indicated by the red trace in Fig. 8. The PRM method clearly required significantly more computation time, especially for large inflation sizes; for this reason, the method was only tested for inflation sizes up to 1000 mm. The increase in computation time with inflation size is due to the increasingly confined spaces within the truss and the correspondingly higher proportion of unfeasible edges in the probabilistic roadmap. By comparison, the proportion of unfeasible edges in the deterministic roadmap only changes slightly.

The lengths of the paths generated by the DRM and PRM algorithms are quite comparable and there is no clear dependency of path length on inflation size. On the other hand, the path length depends strongly on the randomly selected start and end points for the 50 runs.

During the simulations we further observed that the deterministically computed NPs produce relatively shorter paths when the start and points are close to a joint. Whenever a beam obstructs a candidate path, this beam can only be circumnavigated at a joint, requiring additional transit distance that is reflected in the total path length.

We note that a roadmap which was generated for a previous inspection can be saved and re-used for any subsequent planning task, regardless of the start and goal points, resulting in computational savings.

V. CONCLUSIONS

There is increasing interest in methods for robotic infrastructure inspection with the aim of lowering the risk to human inspectors. We have considered the problem of path planning for a small, multirotor UAS inspecting a truss bridge. The proposed algorithm efficiently computes a path through a truss bridge structure from a start point to a goal point. To support path planning, a method was developed to represent the structure in a way that ensures the resulting paths allow for the vehicle's physical dimensions as well as position disturbances and uncertainty. The method represents each structural member of the bridge as a a cuboid, referred to as a beam, which completely contains the member. Points where beams are connected are called joints. Given the necessary parameters, such as joint locations and beam dimensions, a computational representation of the structure is constructed and a set of deterministic navigation points (NPs) are computed around the joints. These NPs are used to define a roadmap. It is assumed that any edge connecting the nodes in this graph is a feasible path segment, that is, a segment which does not intersect the structure. An iterative process of graph search, using the A* algorithm and collision checking, then determines a feasible path.

Simulations performed for an actual bridge span illustrate the functionality of the algorithm and a comparison with a PRM approach indicates its relative efficiency. Specifically, for the scenario considered, the proposed algorithm computed paths of similar path length in significantly less time than the PRM approach.

In future work, we plan to extend the algorithm to plan complete inspection paths that visit a sequence of user-specified inspection points, corresponding to desired vantage points for structural elements requiring inspection.

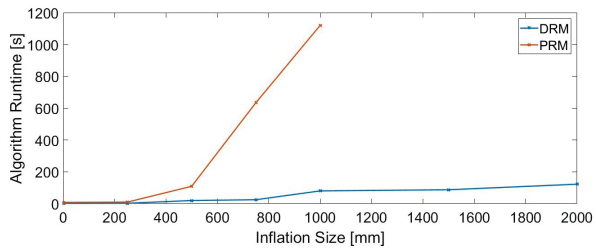


Fig. 8: Runtime of the DRM algorithm over inflation size.

APPENDIX

The mathematical computation of NPs is based on two beams connected at a joint. To compute all NPs for a given structure, the computation is performed for every active joint and for every combination of two active beams at the joint.

In the first step, the plane defined by the centerlines of the two intersecting beams is computed. This plane is also defined by the position of the current joint $\mathbf{p}_{\text{joint}}$ and the plane's normal vector $\mathbf{n}_{\text{plane}}$, obtained in (8). The vectors $\hat{\mathbf{w}}_{\text{b1}}$

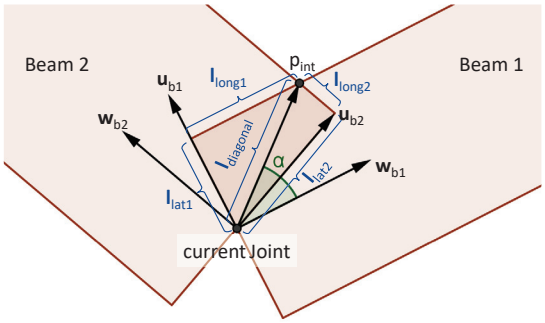


Fig. 9: Vectors and variables for the computation of NPs: Plane cutting the two beams with variables for computing the intersection point $\mathbf{p}_{\text{int}}$.

and $\hat{\mathbf{w}}_{b2}$ represent the direction of the beams' centerlines and point away from the current joint. The direction $\hat{\mathbf{w}}_b$ is equal to the z-axis $\hat{\mathbf{z}}_b$ of the beam's frame if the current joint is the beam's start joint and equal to $-\hat{\mathbf{z}}_b$ if the current joint is the beam's end joint. Fig. 9 shows the connection of two beams viewed from the side. The vectors and lengths used for the computation are marked.

Next, the vectors $\hat{\mathbf{u}}_{b1}$ and $\hat{\mathbf{u}}_{b2}$ in Fig. 9 are obtained:

$$\hat{\mathbf{u}}_{b1} = \hat{\mathbf{w}}_{b1} \times \hat{\mathbf{n}}_{plane} \quad (6)$$

$$\hat{\mathbf{u}}_{b2} = \hat{\mathbf{w}}_{b2} \times \hat{\mathbf{n}}_{plane} \quad (7)$$

where

$$\mathbf{n}_{plane} = \hat{\mathbf{w}}_{b1} \times \hat{\mathbf{w}}_{b2} \quad (8)$$

If their dot product with $\hat{\mathbf{w}}$ from the other beam is negative, the vectors obtained from (6) and (7) must be inverted to ensure that $\hat{\mathbf{u}}_{b1}$ and $\hat{\mathbf{u}}_{b2}$ point in the direction shown in Fig. 9.

To obtain the point $\mathbf{p}_{int}$ where the two beams intersect, the lateral and longitudinal lengths l_{lat1}, l_{long1}, l_{lat2}, and l_{long2} from Fig. 9 must be found. To do so, the lateral lengths l_{lat1} and l_{lat2} are first computed and then a linear equation is solved for l_{long1} and l_{long2}.

The length l_{lat} of a beam is defined as the distance of a tangent, which is perpendicular to the search direction $\hat{\mathbf{u}}_b$ and the beam direction $\hat{\mathbf{w}}_b$. This is shown in Fig. 10. In (9) the search direction $\hat{\mathbf{u}}_b$ is decomposed into the vectors $\hat{\mathbf{x}}_b$ and $\hat{\mathbf{y}}_b$ which are the frame vectors of the beam's frame:

$$\hat{\mathbf{u}}_b = a\hat{\mathbf{x}}_b + b\hat{\mathbf{y}}_b \quad (9)$$

The signs of the scaling coefficients a and b determine the direction of the next vertex:

$$\mathbf{d}_{vertex} = \operatorname{sgn}(a)\hat{\mathbf{x}}_b \left(\frac{x_{size}}{2} + \operatorname{sgn}(a)x_{offset}\right) + \operatorname{sgn}(b)\hat{\mathbf{y}}_b \left(\frac{y_{size}}{2} + \operatorname{sgn}(b)y_{offset}\right) \quad (10)$$

The length l_{lat} is then

$$l_{lat} = \|\mathbf{d}_{vertex}\| \cos(\beta) = \mathbf{d}_{vertex} \cdot \hat{\mathbf{u}}_b \quad (11)$$

The longitudinal lengths l_{long1} and l_{long2} are computed from

$$l_{lat1}\hat{\mathbf{u}}_{b1} + l_{long1}\hat{\mathbf{w}}_{b1} = l_{lat2}\hat{\mathbf{u}}_{b2} + l_{long2}\hat{\mathbf{w}}_{b2} \quad (12)$$

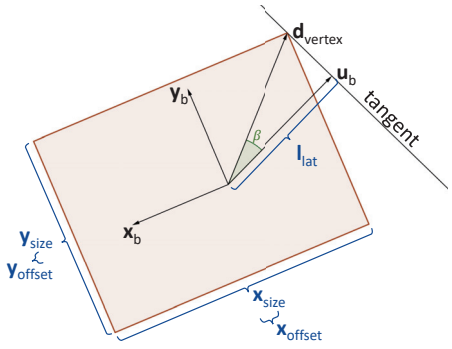


Fig. 10: Vectors and variables for the computation of NPs: Cross-sectional area of a beam and variables for the computation of the extension in direction $\mathbf{u}_b$.

After solving this linear equation, the intersection point is

$$\mathbf{p}_{int} = \mathbf{p}_{joint} + l_{lat1}\hat{\mathbf{u}}_{b1} + l_{long1}\hat{\mathbf{w}}_{b1} \quad (13)$$

Usually two NPs are set for every pair of beams. The NPs are set on the line of intersection which is defined by the intersection point $\mathbf{p}_{int}$ and the plane's normal vector $\hat{\mathbf{n}}_{plane}$. The position along the line is chosen so that the NPs lie on the sides of the beams. This enables to plan paths along the beams. Therefore, the NPs are set based on the maximal extension of the beams in the two directions $\hat{\mathbf{n}}_{plane}$ and $-\hat{\mathbf{n}}_{plane}$. These extensions are denoted as $l_{side1,n}$ and $l_{side1,-n}$ for the first beam and $l_{side2,n}$ and $l_{side2,-n}$ for the second beam and again determine the distance of a tangent. The extensions are obtained in the same way as l_{lat} but this time in the search-direction $\hat{\mathbf{n}}_{plane}$ and $-\hat{\mathbf{n}}_{plane}$. The NPs are then given by

$$\mathbf{p}_{NP1} = \mathbf{p}_{int} + \hat{\mathbf{n}}_{plane} \max\{l_{side1,n}, l_{side2,n}\} \quad (14)$$

$$\mathbf{p}_{NP2} = \mathbf{p}_{int} - \hat{\mathbf{n}}_{plane} \max\{l_{side1,-n}, l_{side2,-n}\} \quad (15)$$

If the two beams are aligned, their centerlines do not define a unique plane. In this case, four NPs are set at the outer edges of the beams, as shown in Fig. 3c.

In the first step, the extensions of the beams are obtained by searching both beams for the tangent distance in the directions $\hat{\mathbf{x}}_{b1}$, $\hat{\mathbf{y}}_{b1}$, $-\hat{\mathbf{x}}_{b1}$, and $-\hat{\mathbf{y}}_{b1}$ by the above described method. The vectors $\hat{\mathbf{x}}_{b1}$ and $\hat{\mathbf{y}}_{b1}$ are the frame vectors of the first beam. Afterwards, the maximum values of both beams are determined for each of the four directions and the NPs can easily be computed.

ACKNOWLEDGMENT

The authors gratefully acknowledge travel support from Virginia Tech's Institute for Critical Technology and Applied Science and from the Center for Unmanned Aircraft Systems under NSF Grant Nos. IIP-1539975 and CNS-1650465.

REFERENCES

[1] R. Bohlin and L. E. Kavraki. Path Planning Using Lazy PRM. In *Int. Conf. Robotics and Automation (ICRA)*, volume 1, pages 521–528, 2000.

[2] P. Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, volume 73. Springer, 2011.

[3] E. E. Hartquist, J.P. Menon, K. Suresh, H. B. Voelcker, and J. Zagajac. A computing strategy for applications involving offsets, sweeps, and Minkowski operations. *Computer-Aided Design*, 31(3):175–183, 1999.

[4] H. Hexmoor. Essential principles for autonomous robotics. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(2), 2013.

[5] S. M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, 2006.

[6] A. Mazumdar and H. H. Asada. Mag-foot: A Steel Bridge Inspection Robot. In *Int. Conf. Intelligent Robots and Systems (IROS)*, pages 1691–1696, 2009.

[7] N. H. Pham and H. M. La. Design and implementation of an autonomous robot for steel bridge inspection. In *Allerton Conf. Communication, Control, and Computing*, pages 556–562, 2016.

[8] P. H. Pignon, T. Hasegawa, and J.P. Laumond. Optimal obstacle growing in motion planning for mobile robots. In *Int. Conf. Intelligent Robots and Systems (IROS)*, pages 602–607, 1991.

[9] R. Wang and Y. Kawamura. Development of climbing robot for steel bridge inspection. *Industrial Robot: An International Journal*, 43(4):429–447, 2016.

[10] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia. Survey of robot 3D path planning algorithms. *J. Control Science and Engineering*, 2016:5, 2016.