In 7th Conference on the Engineering of Computer Based Systems (ECBS 2021), May 26–27, 2021, Novi Sad, Serbia. ACM, New York

Thou Shalt Not Move

A Visibility-based Emergency Stop System for Smart Industries

Gabriele Capannini
Mälardalen University
Västerås, Sweden
gabriele.capannini@mdh.se

Jan Carlson
Mälardalen University
Västerås, Sweden
jan.carlson@mdh.se

Roger Mellander
ABB Robotics
Västerås, Sweden
roger.mellander@se.abb.com

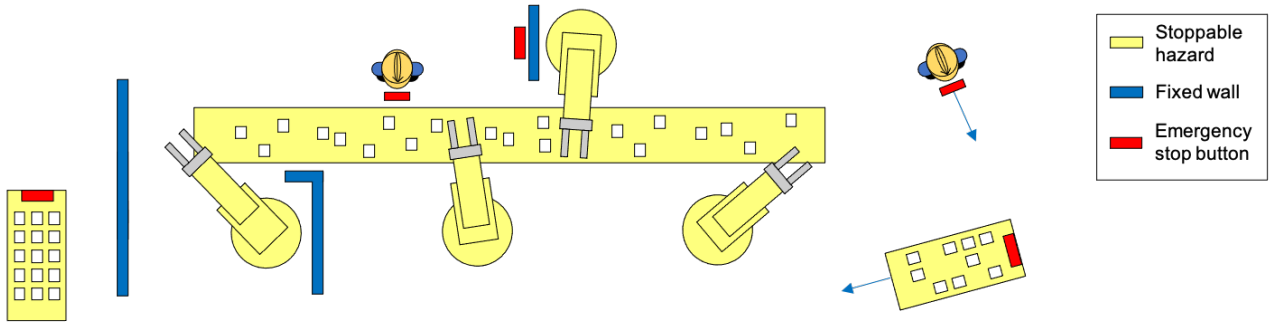

Figure 1: A fence-less production environment with static and autonomous machines collaborating with human operators.

ABSTRACT

Nowadays, industries are crowded with automatized machinery and robots that interact with human operators. In addition to other safety measures already present, we propose a further tool to equip such working places with a visibility-based emergency stop system that only affects those machines that are visible from the position of an emergency stop button when it is pressed. This paper presents the realization of such a system and the preliminary results collected from the conducted scalability experiments.

CCS CONCEPTS

• **Applied computing** → **Industry and manufacturing**; • **Computing methodologies** → **Computer graphics**.

KEYWORDS

Visibility, Emergency stop, Smart industries

ACM Reference Format:
Gabriele Capannini, Jan Carlson, and Roger Mellander. 2021. Thou Shalt Not Move: A Visibility-based Emergency Stop System for Smart Industries. In *ECBS '21: 7th Conference on the Engineering of Computer Based Systems, May 26–27, 2021, Novi Sad, Republic of Serbia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3459960.3459966

ECBS '21, May 26–27, 2021, Novi Sad, Republic of Serbia

ACM ISBN 978-1-4503-9057-6/21/05…$15.00
https://doi.org/10.1145/3459960.3459966

1 INTRODUCTION

The context of this work is a highly automatized factory, where human operators share the space with autonomous vehicles as well as stationary machines. Such a setup naturally comes with a number of safety concerns, including safe routing of autonomous machines, collision avoidance, etc. On top of these mechanisms, however, there is a need for emergency stops that allows human operators to manually stop the equipment if they discover a dangerous situation.

In a traditional factory, with only stationary machines, emergency stops are installed on or near the machines, and it can be statically decided what machines are controlled by each button. In our context, however, some emergency stops are attached to the moving vehicles or are part of the functionality of hand-held devices used by the operators. Even the stationary emergency stops, can not rely only on a static decision of what machines they should control, since they should also stop any moving vehicle nearby.

There is, of course, always the possibility to let every emergency stop control all stationary and moving machines, but in a large factory this can be prohibitively costly. Instead, the starting point for our work has been that an emergency button, when pressed, should stop all machines that are visible from the position of that particular button. From a safety perspective, we can approximate this visibility as long as it is a safe over-approximation, i.e., we can stop a machine even if it is in fact only almost visible, but we must never fail to stop a machine that is, in fact, visible.

The main contribution of this paper is a novel algorithm able to solve the visibility problem with the specific requirements introduced in Section 2. Our algorithm, presented in Sections 3 and 4, has then been used as a building block in the whole emergency stop system described in Section 5. Results from a preliminary evaluation of the visibility algorithm performance are provided in Section 6.

2 THE VISIBILITY PROBLEM

The planar visibility problem is of great relevance in the computer graphics context, e.g., for the removal of hidden lines in 3D scenes, as well as in computational geometry where it finds applications for the shortest-path calculation in presence of obstacles. According to the definition given by Asano *et al.* [3], the problem can be formulated as it follows:

Definition 2.1. Given a set P of *disjoint* polygons and a query point q, define the subset $V(P, q) \subseteq P$ made of all those polygons that are visible from q, where a polygon p is *visible* from q if it exists an open line segment from q to any point of p that does not intersect any other polygon in P.

Such a definition, however, does not fully hold for our scenario since there are some aspects to take into consideration:

(1) we cannot assume that the polygons enclosing the objects are *disjoint*;
(2) due to their conformation and size, some objects let the hypothetical observer placed in q see through them. This contradicts the definition of *visible* polygon stated.

The following sections illustrate among other things, how the proposed solution overcomes these issues.

3 MODELING REAL WORLD OBJECTS

This section describes how the real world objects (e.g., moving robots, machinery, walls, and everything that can affect the computation of a visibility query) are modeled in the proposed solution.

The objects are represented by means of one or more bounding volumes that enclose their plan-view orthographic projection and form the polygons of Definition 2.1. The number and the type of volumes employed to enclose the object depends on how complex the shape to represent is and the accuracy we want to achieve[1]. For example, a machine made of a stationary base and a moving arm can be fully enclosed in one simple box (similarly to Figure 2: low accuracy, better performance) or in a set of tighter volumes: one for each part of it (high accuracy, lower performance). In this preliminary study, any object is represented by a single AABB.

Given a query point q, each bounding volume v is used to calculate the segment $s(v, q)$ representing v from the perspective of q, as exemplified in Figure 2.

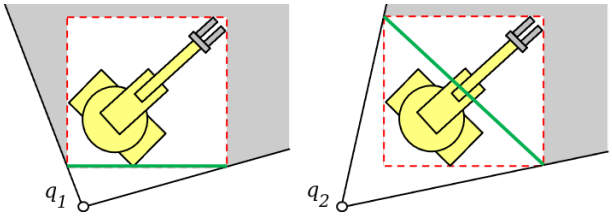

Figure 2: Sketch of how the red AABB v is represented by different segments $s(v, q)$, denoted by green lines, depending on the relative position of q.

[1] In [6], the most popular bounding volumes are described: from the simple Axis-Aligned Bounding Boxes (AABB's) to the more complex Oriented Bounding Boxes.

Observing Figure 2, it is worth noting how bounding the objects in tighter volumes could lead to a more precise calculation of the segments $s(v, q_1)$ and $s(v, q_2)$, thus more accurate solutions.

In addition to the bounding volume and the position, an object representation is also provided of two Boolean attributes denoting:

- **Transparency** — This attribute has been added to address issue (2) reported in Section 2. If the object let an operator see the other objects behind it or if it is possible to see through any part of its bounding volume, the object should be considered transparent. For example, the machine in Figure 2 would be considered transparent, since the corners of the bounding volume can be seen through (and because it is most likely possible to see over or under the "arm").
- **Stoppable** — This attribute states if the object can be stopped or not (e.g., distinguishing between walls and machines).

4 THE VISIBILITY ALGORITHM

Asano *et al.* [3] presented a method to build a data structure (in quadratic time and space) on which it is possible to compute the visibility query in linear time. This solution as well as other approaches presented in [2, 4, 8] are efficient when the visibility problem is solved repeatedly for the same set of polygons. Our context, instead, is highly dynamic and the operations for updating the objects' position are at least as frequent as the visibility query requests. Hence, we opted for storing data in lightweight structures so that object positions can be updated quickly.

Initially, given a query point q, the segment $s(v, q)$ is calculated for each bounding volume v and, then, the endpoints of all segments are sorted according to their polar angle determined in the polar coordinate system where q is the reference point.

Similarly to the approach followed in [3], our algorithm proceeds performing an angular sweep of the sorted endpoints. A ray ρ originated from the query point q is fired towards each endpoint and the segments intersected by ρ are analyzed to discover possibly visible objects. Segments that intersect the ray ρ are stored in a tree[2] T and, unlike Asano *et al.*, our algorithm possibly analyzes all of them. This solves issue (1) reported in Section 2 at the expense of a higher time complexity during the query computation.

As the sweep proceeds, T is kept updated by adding and removing the segment s respectively when the first and second endpoint of s occur. In particular, when s is going to be added to T, we look for a possible non-transparent object in T of which segment is closer to q than s is and, if we found it, we state that s is not visible for the moment. When a segment s is about to be removed from T, no operations are needed if s is transparent (since s does not affect the visibility state of the other objects) while, if s is not-transparent, we search for another non-transparent segment $s' \in T$ of which intersection point with ρ is closer to q than s. If s' exists, removing s from T does not affect the visibility state of the other objects while, if s was the closest segment to q, then we search for the second non-transparent segment s'' closest to q and (if any) all the objects of which segment intersects ρ between s and s'' become visible (if s'' does not exist, then all the objects represented in T are visible).

Figure 3 illustrates the algorithm in a simple scenario. During the sweep, it first adds segment s_1, corresponding to the left wall,

[2] T is implemented as a SuccTree: a lightweight, cache-friendly data structure [5].

and then segment s_2, corresponding to the left machine. At this point, s_2 is hidden by s_1 and thus not visible, but when s_1 is removed from T (at T^3), there is no (non-transparent) segment closer to q than s_2 is, and thus the left machine is visible from q. However, the right machine is not visible from q since there is a non-transparent segment (s_3) closer to q whenever the endpoints of segment s_4 are considered (i.e., at T^6 and T^7).

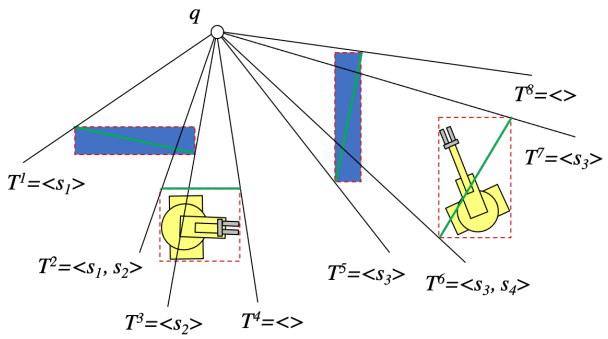

Figure 3: Example of the visibility algorithm.

Let n be the number of objects, the number of segments in T is $O(n)$ hence, according to the SuccTree properties, scanning T costs $O(n \log n)$. Since T is visited at most once for each ray ρ and the rays are $2n$, the time complexity to perform a query with our algorithm is $O(n^2 \log n)$.

5 SOFTWARE ARCHITECTURE

The emergency stop system consists of an *Emergency stop server* and a number of *Emergency stop clients* for the machines and emergency buttons. For simplicity, we assume that the emergency brake system is the only system interested in tracking the position of the moving machines, but in a more realistic setting this would typically be handled by a separate service since positions are also used for, e.g., planning machine movement.

The *Emergency stop server* provides the following services:

- **Register device** — A new device entering the system needs to register with the emergency stop service, including indicating if it is a device that can be stopped and if it has an emergency button.
- **Update position** — A registered device sends its current position and bounding volume (i.e., its AABB) once if stationary, or periodically if it can move.
- **Button pressed** — Informing the server that an emergency button has been pressed.

The *Emergency stop clients* provide the following services:

- **Registration acknowledged** — In reply to the registration, the device receives a unique ID.
- **Stop** — Instructing the device to make an emergency stop.

Since the main non-functional concern is the emergency stop end-to-end latency (i.e., the time from an emergency button being pressed to all visible machines being stopped), we have opted for a solution where overall performance is sacrificed in order to minimize this latency. Thus, rather than computing visibility only when a certain button is pressed, i.e., as part of the **Button pressed** service, the server also contains a periodic activity, **Compute visibility**, in which the visibility is computed for all buttons. When an emergency button is pressed, the systems can directly send a stop message to the machines visible from that button, with very little additional delay.

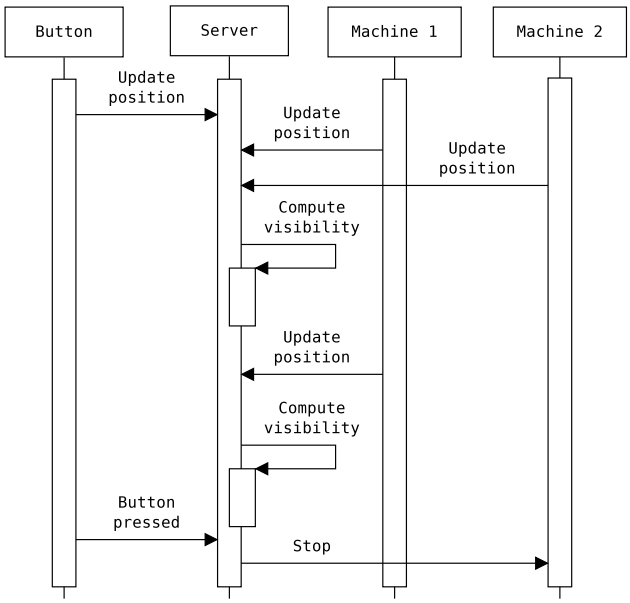

Figure 4: Example of message sequence in a simple scenario.

Figure 4 depicts a simple scenario with a static button and two moving machines (ignoring the registration messages for brevity). The button informs the server of its position once, while the machines update periodically. Here, the update period of the first machine is shorter than that of the second, which would for example be the case if it moves faster. At regular intervals, the server recomputes the visibility information. When the button is pressed, the visible machines (here, only Machine 2) are stopped.

6 EXPERIMENTAL RESULTS

In the perspective of creating the emergency stop system for smart industries, we opted for developing our framework within the Docker ecosystem [7]. We initially built a virtual scenario with two types of Docker containers: *clients* and *server* providing, respectively, the services described in Section 5. To test the system, clients were designed to randomly send messages of different types (i.e., bounding volume changed, position update, button pressed) to the server. The server stores the incoming messages in a queue and processes them one by one, updating position information and sending stop messages to the visible clients as a result of a button being pressed. To ease the creation of such a scenario, the Go programming language was used to implement the Dockers containers, since it offers several facilities for distributed computing [1].

In order to preliminary test the performance and scalability of the proposed visibility algorithm, we also made a stand-alone

C++ implementation of the visibility algorithm, and executed it for scenarios with a varying number (n) of objects and a single query point q. The objects are represented by AABBs with sizes randomly generated within a given range of values according to the uniform distribution. AABBs are placed around q in order to fill the space according to a given density d. Transparent and stoppable AABBs are, respectively, 25% and 50% of the overall n clients.

The results of the preliminary tests are shown in Table 1 and Figure 5. The machine used for these experiments was equipped with an Intel Core i3-1005G1 and 16 GB of memory. Changing the percentage of transparent and stoppable objects in the tests, the variation of the execution time of the algorithm is around 10%.

Table 1: Average elapsed time (in milliseconds) over 150 runs for different values of n and d.

n	$d = 10\%$	$d = 50\%$	$d = 90\%$
100	0.004	0.005	0.005
500	0.019	0.029	0.033
1,000	0.038	0.058	0.062
2,000	0.078	0.113	0.125
4,000	0.166	0.214	0.231
6,000	0.272	0.316	0.344
8,000	0.361	0.417	0.443

Despite the time complexity stated in Section 4, the algorithm performs well in practice and the measured elapsed times grow almost linearly with the input size n even in high-density scenarios as depicted in Figure 5. The performance degrade for dense scenarios is due to the larger number of AABBs stored simultaneously in the SuccTree T, which requires more tests during the scan.

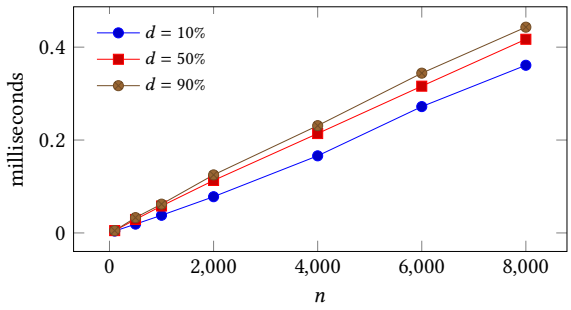

Figure 5: Plot of the experiment results in Table 1.

7 DISCUSSION AND FUTURE WORK

In this paper we have outlined a visibility-based emergency stop system to be used in smart industries where autonomous machines and human operators collaborate.

The dependability of the solution relies on visibility being safely approximated, for example by over-approximating the actual machine shapes by bounding volumes and considering certain objects transparent with respect to not always hiding other objects behind them. In a concrete system instance, there is also a need to consider the relation between the frequency and accuracy of the position updates and the amount of extra over-approximation of the bounding boxes. Finally, from a reliability perspective, it should be pointed out that there is always the possibility of stopping a machine, regardless of the visibility, for example if its position has not been updated recently enough.

The preliminary experiments indicate that the visibility algorithm is sufficiently efficient for use as the basis of a emergency stop systems, for problem sizes up to 8000 objects, which should be more than sufficient for most industrial environments. Although the results from these preliminary synthetic experiments are promising, they need to be complemented by experiments involving the whole system and real scenarios to investigate the overall end-to-end response time including delays caused by the communication between server and clients, in a representative industrial setting.

The proposed solution presents a monolithic system where one server-instance is responsible of the whole service. As a future improvement, we believe that a distributed version where computation can be divided among multiple machines, could lead to several benefits. For example, partitioning the area to be covered among multiple server instances could reduce the communication overhead as well as the workload of the single instances. Moreover, a distributed solution could provide fault tolerance, by dynamically reassigning emergency stop responsibilities to other nodes in case one node is unresponsive.

From the algorithmic point of view we also want to explore how the algorithm can be improved by replacing the query-point q with a query-area, representing the positions from which an operator can reach the button, and then calculate the visibility from any point inside that area. This increases the complexity of the problem significantly, and would mostly likely require some type of approximation algorithm.

ACKNOWLEDGMENTS

This work was supported by the Knowledge Foundation in Sweden through the ACICS project (20190038).

REFERENCES

[1] VN Nikhil Anurag. 2018. *Distributed computing with Go: practical concurrency and parallelism for Go applications.* Packt Publishing Ltd.
[2] B. Aronov, L. J. Guibas, M. Teichmann, and L. Zhang. 2002. Visibility queries and maintenance in simple polygons. *Discrete and Computational Geometry* 27, 4 (June 2002), 461–483. https://doi.org/10.1007/s00454-001-0089-9
[3] Takao Asano, Tetsuo Asano, Leonidas Guibas, John Hershberger, and Hiaroshi Imai. 1986. Visibility of Disjoint Polygons. *Algorithmica* 1, 1 (Jan. 1986), 49–63.
[4] Francisc Bungiu, Michael Hemmer, John Hershberger, Kan Huang, and Alexander Kröller. 2014. Efficient Computation of Visibility Polygons. arXiv:1403.3905 [cs.CG]
[5] Gabriele Capannini and Thomas Larsson. 2018. Adaptive Collision Culling for Massive Simulations by a Parallel and Context-Aware Sweep and Prune Algorithm. *IEEE Transactions on Visualization and Computer Graphics* 24, 7 (2018), 2064–2077.
[6] Simena Dinas. 2019. Bounding Volume Hierarchies for Rigid Bodies. In *Encyclopedia of Computer Graphics and Games*, Newton Lee (Ed.). Springer.
[7] Piotr Dziurzanski, Shuai Zhao, Michal Przewozniczek, Marcin Komarnicki, and Leandro Soares Indrusiak. 2020. Scalable distributed evolutionary algorithm orchestration using Docker containers. *Journal of Computational Science* 40 (2020), 101069.
[8] Gert Vegter. 1990. The Visibility Diagram: A Data Structure for Visibility Problems and Motion Planning. In *Proceedings of the Second Scandinavian Workshop on Algorithm Theory* (Bergen, Sweden) *(SWAT '90)*. Springer, Berlin, 97–110.