Robot Navigation based on an Efficient Combination of an Extended A* algorithm, Bird's Eye View and Image Stitching

Jens Rettkowski, David Gburek, Diana Göhringer
Application-Specific Multi-Core Architectures (MCA) Group
Ruhr-University Bochum (RUB), Germany
jens.rettkowski@rub.de, david.gburek@rub.de, diana.goehringer@rub.de

Abstract— **Robotics combines a lot of different domains with sophisticated challenges such as computer vision, motion control and search algorithms. Search algorithms can be applied to calculate movements. The A* algorithm is a well-known and proved search algorithm to find a path within a graph. This paper presents an extended A* algorithm that is optimized for robot navigation using a bird's eye view as a map that is dynamically generated by image stitching. The scenario is a robot that moves to a target in an environment containing obstacles. The robot is controlled by a Xilinx Zynq platform that contains an ARM processor and an FPGA. In order to exploit the flexibility of such an architecture, the FPGA is used to execute the most compute-intensive task of the extended A* algorithm. This task is responsible for sorting the accessible nodes in the graph. Several environments with different complexity levels are used to evaluate the extended A* algorithm. The environment is captured by a Kinect sensor located directly on the robot. In order to dewarp the robot's view, the frames are transformed to a bird's eye view. In addition, a wider viewing range is achieved by image stitching. The evaluation of the extended A* algorithm shows a significant improvement in terms of memory utilization. Accordingly, this algorithm is especially practicable for embedded systems since they have often only limited memory resources. Moreover, the overall execution time for several use cases is reduced up to a speed-up of 2.88x.**

Keywords—A* algorithm; autonomous robot; path planning; bird's eye view; stitching

I. Introduction

The research area of autonomous robots yields an enormous number of sophisticated challenges. However, this technology holds a lot of promising capabilities for a wide variety of applications nowadays. The applications cross a lot of different domains. For instance, robots can be deployed in hazardous situations instead of humans. They can search for victims in a disaster such as a house collapse [1]. Also robots are often applied in ambient assisted living environments. For example in the RADIO project [2] a robot connected with a smart home environment is presented for an unobtrusive observation of elderly people. This robot analyzes the health conditions of the elderly and provides an intelligent and autonomous help.

In most robotics applications, the robot has the challenges to localize itself in an unknown environment and navigate to a destination without collisions. The interaction with the environment requires several sensors. However, to reduce costs, weight and energy consumption, the number of installed sensors on the robot should be as low as possible. Cameras are often applied to sense the environment. The image data captured from the camera can be used for localization of the robot.

In order to navigate from a robot's start position to a destination, a graph representation of the environment can be used. In order to obtain this representation, a transformation of the camera view is necessary. Graphs are extensively studied for a long time. Hence, approved search algorithms are deployed for path planning. In case of a large search space, the search algorithm can require high memory resources and must provide high performance at the same time. In addition, an autonomous robot is often driven by accumulators. Therefore, the power requirements are highly restricted. Accordingly, a high performance-capable computer with low power consumption must be utilized. An embedded system optimized for specific applications can deal with these demands. In this paper, a combination of dynamic image stitching for a bird's eye view and an extended A* algorithm is applied to robotics. The main contributions of this paper are listed below.

- *Autonomous Robotics.* The work presented in this paper is evaluated by a challenging robotics application. A robot has to navigate autonomously to a destination in an environment containing obstacles. The robot is controlled by a Xilinx Zynq platform that contains an ARM processor and an FPGA. The Zynq processor provides a flexible system for accelerating compute-intensive tasks on the FPGA.
- *Usage of several image processing algorithms for self-localization.* A bird's eye view transformation is conducted to generate a map for path planning. This map simplifies the path planning, obstacle and destination detection. Furthermore, a dynamic image stitching algorithm is applied using a single Kinect sensor. Additional sensors are not needed.
- *An extended A* algorithm.* In order to plan efficiently a path without collisions, the A* algorithm is improved by reducing the size of the search space and the memory utilization. The algorithm is further optimized for robot navigation, by considering also the size of the

robot for the path planning. As a consequence, this extended A* algorithm provides better performance and memory utilization. Furthermore, compute-intensive tasks are executed on the FPGA. The results of this hardware/software codesign are compared against a pure software solution on the ARM processor.

This paper is organized as follows: Section II introduces related work. In Section III the experimental setup with its challenges for the robot navigation is explained. Afterwards, the extended A* algorithm is described in Section IV. Section V presents the evaluation of the complete system. Finally, the conclusion and an outlook are given by section VI.

II. Related Work

In robot navigation, several challenges have to be coped. An autonomous mobile robot must localize itself in an unknown environment. In addition, it has to determine the destination for navigation. Accordingly, the robot is equipped with several sensors. In a lot of cases, a camera acquires images of the environment. These images can be used to build a map for path planning. A bird's eye view of the robot simplifies the localization and map building. A wide variety of image processing algorithms is extensively investigated and is applied in robotics.

In [3], M. Jia et al. present obstacle detection in stereo bird's eye view images for robot navigation. In order to obtain the bird's eye view, a camera system is located at the ceiling of the room. This enables a bird's eye view for indoor navigation of the robot. However, in our approach, the camera is located directly on the robot As a consequence, a transformation is necessary for the bird's eye view, but, our approach is independent from the environment and does not require a specific camera setup in the room. Accordingly, it is not limited to indoor localization.

Another work regarding robot localization using a bird's eye view is presented in [4] by J. Y. Mori et al. An omnidirectional camera is utilized to capture an image. This image presents the complete environment surrounding the robot. Based on this image, the robot localizes itself. The omnidirectional camera has the advantage of a view around the entire robot using only one camera.

In [5], the image of an omnidirectional camera is investigated to distinguish between obstacles and free space by means of a machine learning algorithm. A bird's eye view transformation enables comparison with local occupancy maps.

N. Winter et al. [6] present a robot navigation with visual path planning based on an omnidirectional camera. The transformation of the omnidirectional images into a bird's eye view corresponds to scaled orthographic views of the ground plan. In contrast, our paper presents an approach without the need of an expensive and complex camera system. The robot uses a Kinect that captures images at different angles. These images are stitched to a single image. Digital maps and satellite photos have been built using image stitching algorithms for decades [7]. Several image stitching algorithms have been developed as in [8], [9] and [10]. The algorithm used in this paper is similar to the algorithm in [11].

A multitude of algorithms have been studied extensively and evolved to overcome the challenge of path planning in robotics over the last years. A well-known algorithm is the Dijkstra algorithm. The Dijkstra algorithm determines the shortest path from a source to a destination node in a graph. In contrast to the extended A* algorithm, Dijkstra algorithm deploys an uninformed search. This means that no information is given about the location of the destination. The extended A* algorithm involves the position of the destination. Accordingly, it belongs to the informed searches. They can provide better performance compared to uninformed searches.

An advancement of the Dijkstra algorithm is the original A* algorithm [12]. It reduces the size of the search space due to a heuristic. As a result, the memory resources can be decreased. The extended A* algorithm carries out the same approach and tries to further reduce the memory resources. Another algorithm that optimizes the A* algorithm is the dynamic A* algorithm (D*) [13]. An advantage of this algorithm is that it can reschedule a path when new information about the environment is discovered. The path is repaired incrementally depending on the robot's state. The Field D* algorithm [14] completes this approach. Dynamic changes of the environment also influence the costs between two points that a robot can traverse. NASA's curiosity rover navigates autonomously through the rocky terrain of Mars using the Field D* algorithm. The D* Lite algorithm [15] is also an incremental search as D* or Field D*. It uses heuristics to plan a path and reuse information from previous searches.

Firstly, the search algorithm presented in this paper plans the path before the robot starts to move Afterwards, the robot navigates along the path. Dynamic changes of the environment causes a recalculation of the path with the extended A* algorithm. In this manner, the extended A* algorithm can also reschedule the path. In addition, the processor needs only a single completion of the search algorithm in best case.

A similar approach is developed by H. Liu et al. in [16]. The robot navigation is performed by a hybrid path planning strategy. This strategy plans the path offline with an algorithm called Floyd [17]. In contrast to the extended A* algorithm, a Dijkstra algorithm is executed to select an alternative path when the original calculated path is not available during navigation. However, the extended A* algorithm provides a better memory utilization as mentioned.

III. Experimental Setup & Challenges

In order to show the benefits of the combination from the extended A* algorithm and image processing, a robot application is used as evaluation. The robot is a turtlebot using a Kinect sensor as shown in Fig. 1. The color camera from the Kinect sensor is utilized to capture images from the environment. Indeed, a Kinect sensor is not stringently required. It can also be exchanged with another camera.

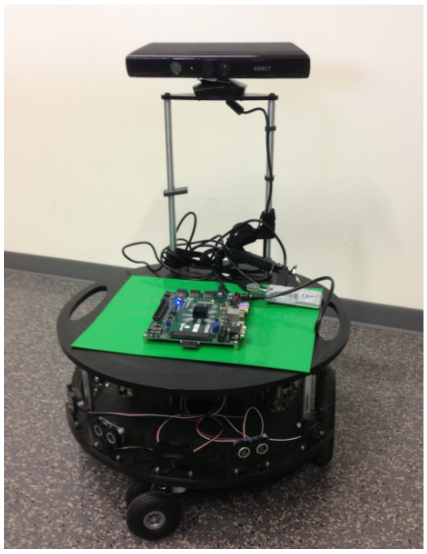


Fig. 1: Robot platform to evaluate the algorithm presented in this paper

Further sensors of the Kinect sensor are not deployed in this work. However, the number of different Kinect sensors provides a good basis for future work.

In this work, the robot is controlled by a Zedboard. The Zedboard contains a Xilinx Zynq chip that has a dual cortex-A9 ARM processor running at 667 MHz. Furthermore, the Zynq chip has a programmable logic that is located besides the ARM processor.

Contrary to a common computer, an advantage of the Zedboard is the lower power consumption. In addition, the heterogeneous architecture of the Zynq chip provides a flexible system for robot application.

The robot is located in an environment consisting of a single target and several obstacles. The robot plans autonomously a path from its start position to the destination. Afterwards, the robot navigates to the destination.

In order to find a path, search algorithms such as an A* algorithm can be applied. These algorithms require a search space presented in a graph structure in order to determine the path. The search space can be constructed using the color image from the Kinect sensor. For that reason, the Kinect sensor captures the environment from the robot's view. However, a search algorithm expects another view of the environment. Therefore, the frames are transformed to a bird's eye view. Another problem is the restricted view of the robot. The robot uses a single Kinect sensor that has a limited viewing range. Hence, the viewing range has to be extended to detect targets at a wider spot. To achieve this, a stitching algorithm is used.

IV. EXTENDED A* ALGORITHM

A lot of applications are existing that try to find a best solution among multiples. The search space is spanned by states of the application.

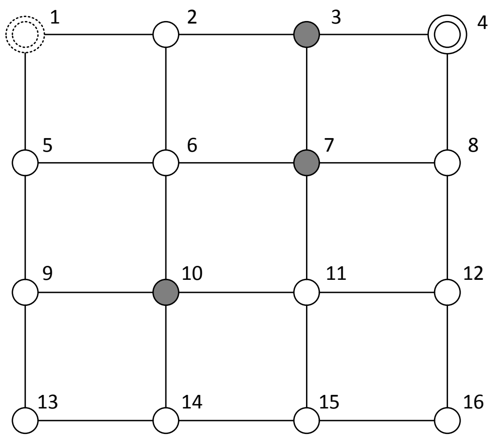


Fig. 2: Graph consisting of a source node 1(two dotted circles), a destination node 4 (two circles, walkable nodes in white and impassable nodes in grey

In this work, the focus is on path planning for a robot. Therefore, an image that presents the environment will be analyzed to find a solution from a start to an end position. The image consists of pixels which forms the search space as a graph.

A. Introduction to A* Algorithm

An example of a graph presentation for path planning is given by Fig. 2. A graph ς contains a set V of nodes and a set E of edges linking the nodes. In case of the robot application, each node represents a pixel captured from the environment.

$$\varsigma = (V, E) \tag{1}$$

In this work, only undirected graphs are investigated. Generally, the nodes are numbered from 1 to n.

$$V = \{1, 2, ..., n\} \tag{2}$$

In Fig. 2, 16 nodes are depicted. An edge ε connects two nodes i and j. The number of the nodes can be used to specify the edge.

$$\varepsilon = (i, j), \quad i, j \in V \tag{3}$$

Every edge *(i, j)* has a value $c_{i,j}$ that determines the cost between the nodes linked by the respective edge. In this paper, the cost $c_{i,j}$ is defined by the distance between nodes. All these costs $c_{i,j}$ are equal since the distance between neighboring pixels is also assumed as identical. A path P from node A to node B can be determined by a series of edges. The end node of every edge is equivalent to the start node of the following edge.

$$\begin{gathered} P(A,B) = \{(i_0, i_1), (i_1, i_2), ..., (i_{k-1}, i_k)\} \\ i_0 = A, i_k = B \end{gathered} \tag{4}$$

B. Sequence of A* Algorithm

In order to find a path from a source node to a destination node, algorithms such as A* can be applied. A* algorithm is an extension of the Dijkstra algorithm. Similar as the Dijkstra algorithm, the A* algorithm finds the path with lowest costs, if a path exists. Contrary to Dijkstra, the A* algorithm conducts the path planning with a heuristic that provides better performance.

The flow of the A* algorithm is explained by an example depicted in Fig. 2. Nodes highlighted in gray are obstacles and cannot be used to find a path. The task is to find a path P(1,4) from node 1 to node 4 without passing the gray nodes. In order to perform the algorithm, an open list L_o for nodes that have to be investigated and a closed list L_c for nodes that are already investigated are defined. Initially, both lists are empty.

$$L_o=\{\nu_0, \nu_1, ..., \nu_\nu\} \quad \nu \in V \tag{5}$$
$$L_c=\{\nu_0, \nu_1, ..., \nu_\nu\} \quad \nu \in V \tag{6}$$

The first step of the A* algorithm is to put the start node into L_o. In case of the example, node 1 is sorted into L_o while L_c is empty.

$$L_o=\{1\} \tag{7}$$
$$L_c=\{\} \tag{8}$$

Afterwards, the A* algorithm takes the first node from L_o. Since only node 1 is in L_o, this node becomes active and will be investigated in the next step. Thereby, neighboring nodes that are directly connected by an edge to the active node are sorted into L_o. In Fig. 2, node 2 and 5 are linked by edges (1, 2) and (1, 5) to node 1. In order to organize the list L_o, the A* algorithm calculates the cost $f(\nu_\iota)$ for every neighboring node ν_ι. The node with the least costs $f(\nu_\iota)$ is at the first position of L_o. The remaining nodes are sorted in decreasing order. The cost $f(\nu_\iota)$ are composed by the addition of the costs $g(\nu_\iota)$ and $h(\nu_\iota)$.

$$f(\nu_\iota)=g(\nu_\iota)+h(\nu_\iota) \tag{9}$$

The function $g(\nu_\iota)$ is the sum of the costs $c_{i,j}$ of all previous edges from the start node to node ν_ι. If the costs $c_{i,j}$ represents the distance, the function $g(\nu_\iota)$ is the distance from the start node to node ν_ι. In contrast to this, function $h(\nu_\iota)$ is a heuristic to predict the costs from node ν_ι to the end node. Hence, the algorithm needs information about the location of the end node. Since the position of the end node is given by the undirected graph, a heuristic can be used to estimate the costs from a node to the end node. In this paper, the Manhattan distance is utilized as the heuristic $h(\nu_\iota)$. This function adds the costs of every horizontal and vertical edge from node ν_ι to the end node. Thereby, the minimal path cost is calculated without the inclusion of obstacles.

This organization of the list L_o assures that the most promising node is investigated first. Since $f(5) = 5$ and $f(2) = 3$, L_o is ordered as in Equation (10). Node 1 is closed and is moved to list L_c.

$$L_o=\{2, 5\} \tag{10}$$
$$L_c=\{1\} \tag{11}$$

Afterwards, the steps are repeated. The first node of L_o is node 2. The algorithm analyzes the available edges from node 2. The edge (2, 3) points to an obstacle. That is why node 3 is ignored. Only node 6 with $f(6)=5$ is reachable and is sorted in L_o. Node 2 is moved to list L_c.

TABLE I. A* PSEUDO CODE FOR ROBOT PATH PLANNING

```
1: initialize the list $L_o$
2: initialize the list $L_c$
3: start node = A
4: end node = B
5: sort_$L_o$(start node)
4: while $L_o$ is not empty
5:   active node = first_$L_o$
6:   insert_$L_c$(active node)
7:   find_neighbors(active_node)
8:   for each neighbor
9:        if neighbor = end node → stop search
10:       if neighbor = gray →skip this neighbor
11:       if neighbor is already in $L_o$ →
12:                if g(neighbor) < g(node in $L_o$) → exchange
13:       otherwise sort_$L_o$(neighbor)
14:   end for
15: end while
```

$$L_o=\{6, 5\} \tag{10}$$
$$L_c=\{1, 2\} \tag{11}$$

Node 6 is at the first position of L_o. Depending on the application, node 6 can also be listed after node 5 as both have the same costs $f(\nu_\iota)$. When a node directs to a node that is already in the list L_o, the costs $g(\nu_\iota)$ from both nodes are compared.

The node with lower costs $g(\nu_\iota)$ is placed into the list L_o. This procedure is repeated until the end node is detected or the list L_o is empty. The latter case occurs when no path exists from start to end node. If a path is found, the A* algorithm using the Manhattan heuristic computes the minimal path. In order to construct the path, every node stores the source node that points to this node. When the end node is found, the path is constructed by retracing the source nodes. A pseudo code for the A* algorithm is shown in Table I. The A* algorithm has a time complexity of $O(|V^2|)$ assuming that the list L_o is implemented as a binary heap and list L_c as an array. Furthermore, the heuristic must be monotone.

C. Sequence of extended A* Algorithm

As mentioned the A* algorithm finds an existing minimal path. Consequently, a robot using this algorithm is able to plan a path to its destination. However, the list L_o collects all nodes that construct the path to the destination and further nodes that are discarded later. Depending on the application, this list can contain a tremendous number of nodes. Line 13 of the pseudo code in Table I shows that each new node must be placed in list L_o. The new node has to be placed in order, since the algorithm uses the first node of L_o in every iteration. This organization is expensive in terms of computation time for a large list. There are several ways to optimize the organization by using more efficient sort algorithms. This paper follows another approach to reduce the computation time of the sorting. The extended A* algorithm keeps the list L_o small. Consequently, the computation time for sorting is reduced. In addition, this reduction is independent from the sorting algorithm. Furthermore, a large list can overload the system memory. In case of embedded systems, especially memory is a limited resource. Therefore, the list should be kept small to avoid system problems. Assuming the A* algorithm searches for new nodes as it is written in line 7 in Table I, the new node will be sorted in list L_o when the following criteria are satisfied.

1. *End node. The new node is not the end node. If it would be the end node, a path is found. As a result, the algorithm terminates.*
2. *Permitted node. The new node is not a permitted node. This is a criterion given by the robot application. A node highlighted gray in the graph is equivalent to an obstacle. Hence, it cannot be included in the final path.*

This paper extends the A* algorithm with a further criterion that minimizes the list length. The new criterion is called Memory Criterion.

3. *Memory Criterion. A new node is only added to the list L_o when nodes in a defined distance d are not gray.*

This criterion ensures that not only the neighbor node is analyzed. A distance d specified in number of nodes is defined. If at least one node that is d nodes away from the neighbor node is gray, the neighbor node is not inserted into the list L_o. Thus, the list size is more constrained and according to this, the number of nodes is reduced. In case of the robot application, this criterion does not exclude possible paths for the robot. The robot has a width given by its dimension which can be used to define the distance d. This distance can be considered as a safety distance to avoid collision with an obstacle. As a result, the memory criterion improves the memory utilization and additionally, it has practical reasons. Furthermore, a neighbor node that is already in L_o will be excluded from the list. This reduces further the memory resources. Assuming that the map has a large size as it is often the case and including the robot size, this exclusion has no significant impact on the success of the robot navigation. Table II shows the pseudo code of the extended A* algorithm. The memory criterion is located in line 10. The second criterion is written in line 11. This line is still necessary, since gray nodes that are located in a range smaller than the distance d after the start-up of the application can be inserted into the list L_o without this criterion. Another way to avoid a high number of nodes inside the list is to cluster nodes. Afterwards, these clusters are the new elements of the list. However, this approach is not comparable with the extended A* algorithm. The resolution decreases as well as the number of elements. The extended A* algorithm keeps the resolution while decreasing the number of nodes inside the list.

V. Robot Pathplanning

As mentioned, the extended A* algorithm is implemented for path planning of a robot. The code is written in C++ and the Zedboard is equipped with a Linaro operating system since Linaro provides device drivers for several peripherals. Images of the environment are acquired with the Kinect sensor that is accessible by the OpenNI 2.0 drivers. Three experimental setups consisting of a target and multiple obstacles are constructed for evaluation. In order to distinguish between target and obstacles, the algorithms analyzes the color of the objects. The color of targets is defined as green. The color purple is used for obstacles.

Table II. Extended A* pseudo code for robot path planning

```
1: initialize the list $L_o$
2: initialize the list $L_c$
3: start node = A
4: end node = B
5: sort_$L_o$(start node)
4: while $L_o$ is not empty
5:    active node = first_$L_o$
6:    insert_$L_c$(active node)
7:    find_neighbors(active_node)
8:    for each neighbor
9:        if neighbor = end node → stop search
10:       if neighbor doesn't satisfy Memory Criterion → skip
          this neighbor
11:       if neighbor = gray →skip this neighbor
12:       if neighbor is already in $L_o$ →
13:              delete neighbor
14:       otherwise sort_$L_o$(neighbor)
15:   end for
16: end while
```

The entire software implemented on the Zedboard consists of two main components.

1. The image processing of frames includes among others the image stitching and bird's eye view transformation. This is needed for further processing. The functions used for image processing are based on the OpenCV 2.4.11 library.
2. The navigation of the robot using the extended A* algorithm and the control of the robot. The control of the robot performs the navigation based on the path planned by the extended A* algorithm for robotics.

A. Image Processing of Frames

The Kinect sensor acquires images from the environment. However, the viewing range of the Kinect sensor is limited. Hence, it is not possible to transform these images to an entire bird's eye view based on a 360° view. If the robot does not find the destination, it assumes that the destination is not in the viewing range of the Kinect sensor. In order to increase this viewing range, the robot rotates at 45° and captures five times a single image at different angles. These images extend the view of the robot since it scans the environment. In this paper, the robot uses dynamically the stitching algorithm. This adaptive behavior optimizes the performance of the system since image stitching has high computational costs in terms of performance as shown in Section V.

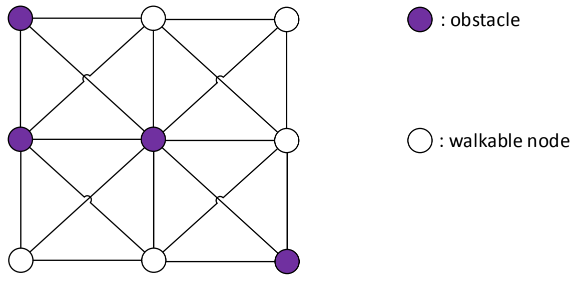


Fig. 3: A graph based on the binary image

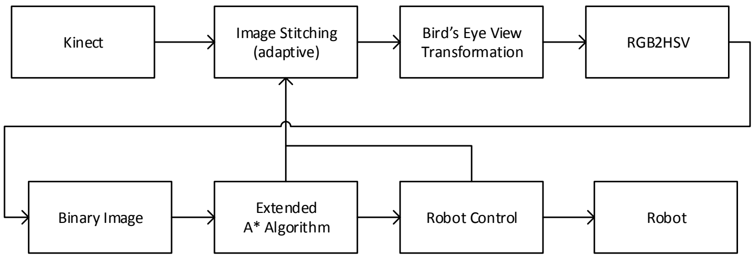


Fig. 4 Algorithms chain of the implemented robot navigation

Multiple images captured during the scan process are joined to a single image for a bird's eye view transformation using the stitching algorithm. In this work, the number of images for stitching is configurable. As a consequence, the execution time of the stitching algorithm depends on the number of images that has to be joined and can be configured to the application-specific requirements. The main parts of the stitching algorithm are finding the features within the images, matching the features of the images and determine the homography matrix to transform the images for blending them. The image resulting from this algorithm is transformed in a top down bird's eye view. In order to perform the bird's eye view transformation, corner points of a trapezoid are defined inside the image. These points are the corner points of the bird's eye view and are configured by the programmer. The distortion correction is indicated by the corner points in the image. By means of these points, the homography matrix is calculated to transform the image in a top down bird's eye view. The transformation dewarps the image captured by the robot and accordingly, it enables a simpler calculation of the robot movements. Each pixel of the image represents a node for the A* algorithm. The pixels are captured in RGB format. Obstacles and destination are detected by analyzing the color of the pixels. Since color recognition is sensitive to changing lighting conditions, the image is converted to HSV color space. An HSV color space provides more robust obstacle detection. Thus, the robot distinguishes between nodes that belong to an obstacle and those, which are walkable. For that reason, the image is converted to a binary image. Each binary pixel is either a node of an obstacle or a walkable node for the robot. Moreover, the use of a binary image reduces the memory resources.

B. *Robot Navigation using the Extended A* Algorithm*

The nodes of the graph that has to be analyzed are represented by the pixels of the binary image. In this work, the nodes are linked. In addition to the horizontal and vertical links, the nodes have diagonal links. This increases the search space for the robot.

However, it also increases the number of paths to the destination. The costs c_{hv} of the horizontal and vertical links are equal in contrast to the diagonal link c_d. This link has a cost which is calculated by the Pythagorean Theorem.

$$c_d = \sqrt{c_{hv}^2 + c_{hv}^2} \tag{12}$$

The binary image is analyzed by the extended A* algorithm described in Section IV. The maintenance of the open list is time-consuming, since it has to be sorted in each iteration. Therefore, the performance of the algorithm is influenced by the implementation of the open list. The open list is implemented with a binary heap structure. In case of a binary heap structure, only the node with the lowest cost is always selected as the new node for investigations. In other words, detailed information about the order of the entire list is not necessary. Thus, a binary heap structure is an efficient and useful implementation for the list in software. Furthermore, a binary heap and another implementation, hereafter referred to as HW sort, to sort the list have been implemented inside the FPGA. The ARM processor communicates via the General Purpose (GP) port to these hardware components. Both of them support four functions to clear the list, add a new element, remove an element and re-sort an element with new costs. These functions are necessary to build the A* algorithm. The binary heap from the software approach is synthesized with Vivado HLS 2014.4. The HW sort synthesized also with Vivado HLS adds an element to the list based on the incoming order. For removing an element, the whole list will be analyzed to find the element with the lowest costs. Afterwards, this element will be removed from the list. The function call to re-sort the list updates nodes with their new costs.

The robot navigates autonomously along a path determined by the A* while it monitors the environment with the Kinect sensor. Dynamic and relevant changes of the environment induce a new start of the program sequence. Obstacles that are occurring in front of the robot are relevant changes. In this case, the robot brakes immediately to avoid a collision. To sum up, the steps that are conducted in this work are illustrated in Fig. 4.

VI. Evaluation

The implemented software on the ARM processor for controlling the robot and for robot navigation has a size of 51 kB. Controlling the robot requires 6 kB of memory, while the robot navigation has a size of 45 kB. The first step of the robot navigation is image stitching. The image stitching algorithm implemented in this paper extends the view from 61.5° to 110°. Consequently, the view is expanded by a factor of 1.8 as shown in Table III. The image stitching function receives an array of input images and uses the SURF algorithm [18] to determine correlated points between the images.

TABLE III. View Extension by Image stitching

Original View	Expanded View	Improvement
61,5°	110°	1.8x

In case of five acquired images around the robot that are forwarded to the SURF algorithm, it has a failure rate of 40%. This means that the algorithm was not able to find common features for all five images. Accordingly, it is not possible to join the images. Since the measurements are varying, the number of measurements was set to 20. It assures reliable results. An implemented modification of the image stitching function is that the SURF algorithm compares only two images at a time. The images that have to be combined are forwarded separately in pairs. This modification influences the performance and failure rate. Due to the separation in pairs, the performance is reduced by 0.09 % as presented in Table IV. However, the failure rate is decreased by the factor of 2. The results are listed in Table V. The degradation of performance is negligible in comparison to the improvement regarding the failure rates. The average execution time for the subsequent bird's eye view transformation of a single frame is presented in Table VI. In order to analyze the extended A* algorithm, three experimental setups are used. Fig. 5 presents an abstract view of these setups. Setup A is the simplest. It contains no obstacles in contrast to setup B and C. Setup C has the highest complexity and most restrictions on the possible path to the target. The image size for the extended A* algorithm is 640x480 pixels and resized down to 320x340 in the software. Fig. 6 presents a comparison between the classical A* and the extended A* algorithm running only in software. The execution time of the extended A* algorithm is up to 2.88x faster compared to the classical one.

TABLE IV: GENERIC PERFORMANCE OF IMAGE STITCHING WITH AND WITHOUT MODIFICATION

	No modification	Modification
Exec. time [sec.]	24,5	27

TABLE V. FAILURE RATE OF IMAGE STITCHING WITH AND WITHOUT MODIFICATION

	No modification	modification
failure rate	40 %	20 %

TABLE VI: AVERAGE PERFORMANCE OF THE BIRD'S EYE VIEW TRANSFORMATION

	Performance per frame [sec.]	Video display in frames per seconds
Bird's Eye View	0.245	≈4

TABLE VIII: RESOURCE UTILIZATION OF THE FPGA-BASED LISTS OBTAINED FROM VIVADO HLS 2014.4

	LUTs	FF	BRAM_18K
Binary Heap	2709	5706	3
HW sort	12753	3401	3

The speedup is achieved due to the lower memory requirements. In Fig. 7 the maximum number of nodes inside the list is shown. The extended A* algorithm provides the smallest list. Therefore, the effort to sort this list is lower resulting in a better performance. The performance results of the different implementations for the extended A* algorithm are shown in Fig. 8. The A* algorithm with an FPGA-based binary heap for the open list shows the worst performance results. The structure of the binary heap cannot be efficiently implemented by Vivado HLS in an FPGA. Therefore, the HW sort for the open list has been implemented using VivadoHLS that shows a significant improvement in comparison to the FPGA-based binary heap. In case of no obstacles, this structure shows the best performance. The resource utilization of both FPGA-based lists running at 200 MHz is given by Table VIII.

FIG. 5: EXPERIMENTAL SETUP TO ANALYZE DIFFERENT IMPLEMENTATIONS OF THE EXTENDED A* ALGORITHM

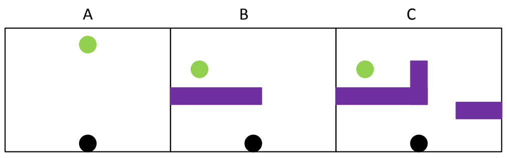


FIG. 6: COMPARISON BETWEEN CLASSIC IMPLEMENTATION AND EXTENDED A* ALGORITHM

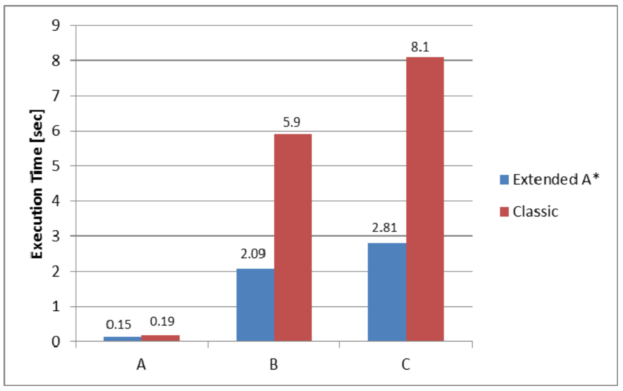


FIG. 7: THE MAXIMUM NUMBER OF ELEMENTS FOR THE EXTENDED A* AND THE CLASSICAL IMPLEMENTATION

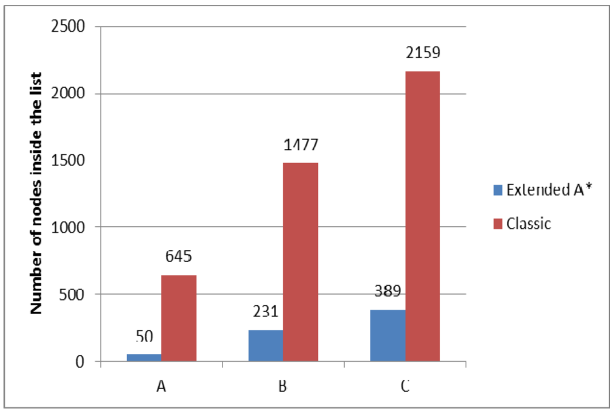
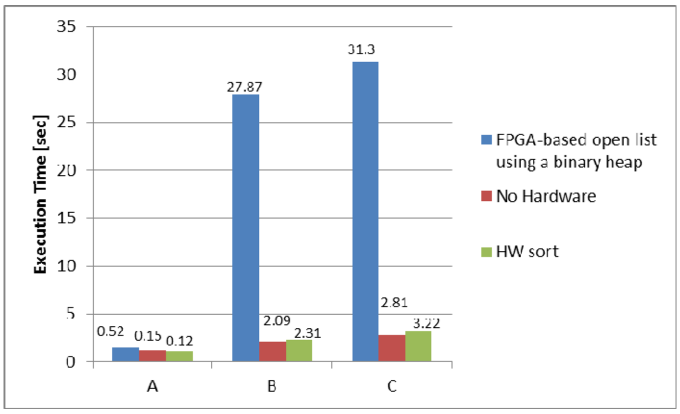


FIG. 8: PERFORMANCE OF EXTENDED A* ALGORITHM USING NO HARDWARE, AN FPGA-BASED BINARY HEAP AND HW SORT



VII. CONCLUSION & OUTLOOK

In this paper, a robot navigation[1] is presented using a combination of sophisticated image processing algorithms to generate a map and an extended A* algorithm to plan the path based. A turtlebot localizes itself in an environment containing several obstacles and a destination. Afterwards, it autonomously navigates to the destination. In order to generate a map, a Kinect sensor is used to scan the environment. If the robot cannot find the destination, it spins around and captures images at different angles. These images are joined to a single image with a wider angle using a stitching algorithm. The stitching avoids the usage of an expensive camera system. Afterwards, this image is transformed to a bird's eye view providing a dewarped view of the robot in its environment. The map given by the bird's eye view is analyzed by an extended A* search algorithm to determine a path to the destination. In contrast to the original A* algorithm, the number of nodes analyzed to plan the path is reduced by a maximum factor of 5.55x in the use cases presented in this paper. Based on this reduction, the memory resources are optimized especially for embedded systems that have limited memory. In addition, the performance is increased by a maximum factor of 2.88 due to the modifications of the A* algorithm. In future work, the goal is to further improve the performance, by outsourcing more tasks to the FPGA.

ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 643892.

REFERENCES

[1] K. Pathak and S. K. Agrawal, "An integrated path planning and control approach for nonholonoic unicycles using switched local potentials." IEEE Transactions on Robotics, vol. 21, no. 6, pp. 1201-1208, 2005.

[2] C. Antonopoulos, G. Keramidas, N. S. Voros, M. Hübner, D. Göhringer, M. Dagioglou, T. Giannakopoulos, S. Konstantopoulos, V. Karkaletsis, "Robots in assissted living environments as an unobtrusive, efficient, reliable and modular solution for independent ageing: the radio perspective", In Proc. of the 11th International Symposium on Reconfigurable Computing: Architectures, Tools and Applications (ARC), Bochum, April 2015.

[3] M. Jia, Y. Sun, J. Wang, "Obstacle detection in stereo bird's eye view images," Information Technology and Artificial Intelligence Conference (ITAIC), 2014 IEEE 7th Joint International , vol., no., pp.254,257, 20-21 Dec. 2014

[4] J. Y. Mori, Janier Arias-Garcia, C. Sánchez-Ferreira, Daniel M. Muñoz, C. H. Llanos, and J. M. S. T. Motta, "An FPGA-Based Omnidirectional Vision Sensor for Motion Detection on Mobile Robots", International Journal of Reconfigurable Computing Volume 2012 (2012), Article ID 148190, 4 April, 2012.

[5] G. E. M. de Buy Wenniger, T. Schmits, and A. Visser, "Identifying free space in a robot bird-eye view", 4th European Conference on Mobile Robots, Minli/Dubrovnik, 23-25 Sep., 2009.

[6] Winters, N.; Gaspar, J.; Lacey, G.; Santos-Victor, J., "Omni-directional vision for robot navigation," Omnidirectional Vision, 2000. Proceedings. IEEE Workshop on , vol., no., pp.21,28, 2000.

[7] D. L. Milgram. Computer methods for creating photomosaics. IEEE Transactions on Computers, C-24(11):1113–1119, November 1975.

[8] Zhen Hua; Yewei Li; Jinjiang Li, "Image stitch algorithm based on SIFT and MVSC," Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on , vol.6, no., pp.2628,2632, 10-12 Aug. 2010.

[9] Senarathne, C.N.; Ransiri, S.; Arangala, P.; Balasooriya, A.; De Silva, C., "A faster image registration and stitching algorithm," Industrial and Information Systems (ICIIS), 2011 6th IEEE International Conference on , vol., no., pp.66,69, 16-19 Aug. 2011.

[10] Zhu Lin; Wang Ying; Zhao Bo; Zhang Xiaozheng, "A Fast Image Stitching Algorithm Based on Improved SURF," Computational Intelligence and Security (CIS), 2014 Tenth International Conference on , vol., no., pp.171,175, 15-16 Nov. 2014.

[11] Brown and D. Lowe. Automatic Panoramic Image Stitching using Invariant Features. International Journal of Computer Vision, 74(1), pages 59-73, 2007.

[12] E. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.

[13] Ferguson, D., and Stentz, A. 2005. The Delayed D* Algorithm for Efficient Path Replanning. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

[14] Anthony Stentz. The Focussed D* Algorithm for Real-Time Replanning. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1995.

[15] S. Koenig and M. Likhachev. D* Lite. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2002.

[16] Hui Liu; Stoll, N.; Junginger, S.; Thurow, K., "A Floyd-Dijkstra hybrid application for mobile robot path planning in life science automation," Automation Science and Engineering (CASE), 2012 IEEE International Conference on , vol., no., pp.279,284, 20-24 Aug. 2012.

[17] Robert W. Floyd: Algorithm 97 (SHORTEST PATH). In: Communications of the ACM 5, 1962, 6, S. 345.

[18] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346--359, 2008.

[1] A demonstration of the running system can be seen on Youtube: https://www.youtube.com/watch?v=x5UG7OSvk58