Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 209–214

DOI: 10.15439/2015F232
ACSIS, Vol. 5

Fast Artificial Landmark Detection for Indoor Mobile Robots

Dmitriy Kartashov, Arthur Huletski
The Academic University
Saint-Petersburg, Russia
{dmakart, hatless.fox}@gmail.com

Kirill Krinkin
St.Petersburg State Electrotechnical University "LETI"
St.Petersburg, Russia
kirill.krinkin@fruct.org

***Abstract*—Nowadays the big challenge in simultaneous localization and mapping (SLAM) of mobile robots is the creation of efficient and robust algorithms. Significant Number of SLAM algorithms rely on unique features or or use artificial landmarks received from camera images. Feature points and landmarks extraction from images have two significant drawbacks: CPU consumption and weak robustness depending on environment conditions. In this paper we consider performance issues for landmark detection, introduce a new artificial landmark design and fast algorithm for detecting and tracking them in arbitrary images. Also we provide results of performance optimization for different hardware platforms.**

I. Introduction

LANDMARKS are generally defined as passive objects in the environment that provide a high degree of localization accuracy when they are within the robot's field of view [1]. Artificial landmarks may carry additional information about the environment and may be used to assist a robot in localization and navigation. Although this approach requires environment preprocessing, it makes developers free to choose a type of landmark and information it holds. Furthermore, a developer could use any design for the landmark that can be sensed in small (with respect to an entire environment) location, but since artificial landmarks are supposed to simplify extraction of location features, appropriate landmark should satisfy the following requirements:

- can be reliably detected in a given environment. Detection should be robust for bad lighting conditions, glares, wide spectrum detection angles;
- can be identified;
- can be easily created and fixed in a given environment.

It this paper we analyze existing approaches for landmarking and introduce new kind of color landmark for fast and robust detection. The paper is organized as follows. In the next section quick response codes [2] and detection methods for them are discussed. In Section III. new design for color landmark is introduced and detection algorithm is suggested. In Section IV. tracking approach is introduced. In Section V. algorithm performance and detection rate are discussed. Finally, in Section VI. a summary of this paper and some issues for further development are listed.

II. Artificial landmarks

One possible option for artificial landmark is a visual printed landmark, e.g. QR codes. This kind of landmarks has several advantages comparing to RFID and other technologies: it doesn't consume power, requires only camera which most mobile robots are already equipped, cheap and easy to produce.

There are several ways to detect QR codes in an image. In [3] the Viola-Jones framework based on Haar features is used to detect QR finder patterns (FIPs). Found FIPs are aggregated into a graph by their size and distance between them. The graph is searched for 3-cycles that satisfy the orientation criterion and represent QR codes. Unfortunately, the QR code plane must be almost orthogonal to the camera axis to reach the claimed 90% detection rate. In addition, this algorithm gives no information about the QR code position in the space.

Another approach that is described in [4] utilizes a special line parametrization called PClines which is a variant of the Hough transform. This parametrization uses a parallel coordinate system and allows faster accumulation than the basic Hough-transform method. The image is searched for the specific parallel lines pattern that is declared as QR code. The algorithm can handle various orientations of QR codes in images, tolerant to uneven illumination and allows real-time processing but is unable to detect QRs from far away or in blurred images. It's also unclear whether the algorithm can detect several QRs on the same image.

Some other algorithms (e.g. [5], [6]) assume that there is only one QR code in the image, so they are not suitable for our problem as the robot may see several landmarks simultaneously.

In order to estimate performance and resource requirements a variant of the QR detection algorithm described in [3] have been implemented. It uses cascade classifier to find both the finder and alignment patterns in the detected QR code. The last is necessary since 4 points are needed in order to compute position of the landmark in the space, but the existing algorithm gives only 3 points. Experiments have shown that the usage of bare QR codes as landmarks has following significant drawbacks:

- it's rather difficult to detect and extract bare QR code when it is far enough (more than 1 meter, which is quite often condition for indoor mobile robots);

978-83-60810-66-8/$25.00

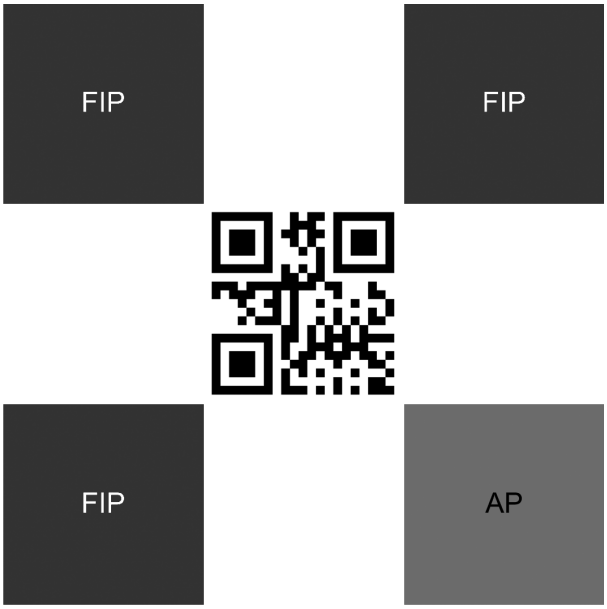


Fig. 1. Landmark design

- detection quality turns out to be very sensitive to the angle between camera and QR code plane.

So it was decided to create a new type of landmark that can be easily and reliably detected (i.e. chance of detection doesn't depend much on the camera position) and allows to carry extra information (like position or point identifier in the environment, or even instructions for mobile robots).

Various types of such visual landmarks are discussed in [7]. Besides good discussion of existent landmark types, this work introduces a landmark design based on QR code. According to it QR code is placed into blue rectangle that has three colored circles around it. Landmark of proposed design is scalable, special algorithm is introduced to extract inner area from outer circle for further QR code detection. Provided evaluation claims that landmark with diameter of outer circle equal to 20 cm can be reliably detected from 2 meters while its horizontal angle lies in range $[-60°; 60°]$.

The aim of our work is to propose a landmark that can be detected in broader horizontal angle range and has smaller size at the same time. The novel design was inspired by QR code detection algorithm described in [3] and [8]. The last one introduces artificial striped landmark. Two colors are used for stripes, each stripe has neighbours of different color.

III. Landmark design and detection

In this section the new artificial landmark design and detection algorithm are described.

A. Landmark layout

A general idea for a new landmark is based on QR code layout extended with color markup. The suggested layout consists of 3 blue squares called finder patterns (FIP) and one red square – alignment pattern (AP) – on the white background (in fact, any 2 easily distinguishable colors may be used). They are located on the landmark like FIPs and APs in QR codes (see Fig. 1). The significant feature of such layout is that the landmark looks like 4 light squares on a dark background in the saturation channel of the HSV color space in daylight (Fig. 3b). This allows to run some edge detection algorithm on the saturation channel to find contours in the image (Fig. 3c) and some of the detected contours are chosen to be FIP or AP candidates for the landmark.

B. Detection algorithm

The full algorithm pipeline is shown in Fig. 2. Below we discuss each stage of the algorithm.

To select FIP candidates consider bounding rectangles of the detected contours in the RGB color space. For each candidate all pixel values within the bounding rectangles are accumulated separately for each channel. Then the following conditions should be checked:

$$\sum blue > a \cdot \sum red \quad \text{and} \quad \sum blue > b \cdot \sum green$$

If these conditions are satisfied then the contour is pushed into the list of FIP candidates. If the similar conditions are met for the red channel then the contour is stored as an AP candidate. The coefficients a and b in the formula above are called a color ratio and determine significance of the color component within the contour bounding rectangle: higher coefficient values discard more candidate contours at this stage. On the other hand, the farther from the camera the landmark is the lower the coefficient values should be to detect a FIP. The coefficient values from the range $[1.2; 1.6]$ are reasonable to use in practice. The output of this stage is shown in Fig. 3d.

Note that the found contour is not checked to be rectangular: due to optical or perspective distortions the specific shape of a FIP might be hardly recognizable. So there is no constraints

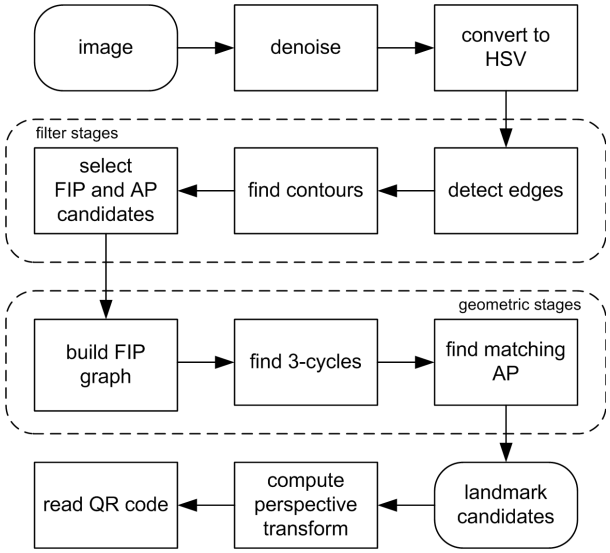
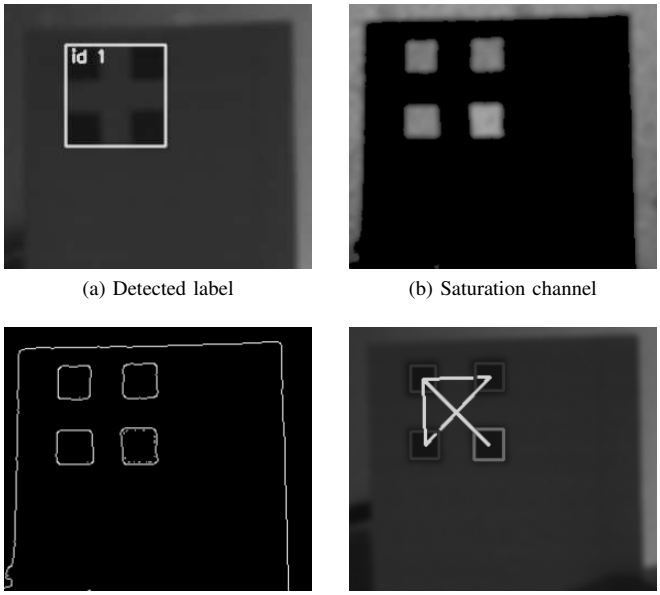


Fig. 2. Detection algorithm pipeline



(a) Detected label (b) Saturation channel

(c) Edge detector output (d) Detected FIPs and APs

Fig. 3. Detector output

on the landmark shape at all. In our case, discarding contours by color is more robust than by geometric features, though this method is very sensitive to the external lightning and FIP selector parameters should be adjusted depending on the environment conditions.

After obtaining the lists of FIPs and APs quadruples (3 FIPs and 1 AP) are formed from candidates using landmark geometric features. At first, the graph is constructed in the following way: vertices are FIPs and an edge connects FIPs if they satisfy two types of constraints:

1) the minimum and maximum distance between FIPs:

$$\begin{cases} D_{min} \cdot width < |X_{FIP1} - X_{FIP2}| \\ D_{min} \cdot height < |Y_{FIP1} - Y_{FIP2}| \\ |X_{FIP1} - X_{FIP2}| < D_{max} \cdot width \\ |Y_{FIP1} - Y_{FIP2}| < D_{max} \cdot height \end{cases} \quad (1)$$

where $width$ and $height$ are the size of a FIP pair; X, Y – FIP position; D_{min}, D_{max} – constraint coefficients;

2) the difference in height and width:

$$\begin{cases} W_{min} < width_{FIP1}/width_{FIP2} < W_{max} \\ H_{min} < height_{FIP1}/height_{FIP2} < H_{max} \end{cases} \quad (2)$$

where $width$ and $height$ – FIP size; H, W – size ratio coefficients.

Parameters in these constraints may vary and their actual values depend on the layout of the landmark. In our case a distance constraint is from 1 to 3 FIP sizes and a FIP size ratio is from the range $[0.5; 1.5]$. The resulting graph is searched for 3-cycles that are considered to be landmark candidates.

Finally, the last component of the landmark is added – AP. From all of the AP candidates we choose one that meets the constraints on the location (the top-left corner or center must be inside the bounding rect of the three selected FIPs) and which size is closest to the selected FIPs. Note that the constraint on the top-left corner doesn't allow detecting upside down landmarks. If an appropriate AP is found then the landmark candidate is accepted.

The output on each stage of detection is shown on the Fig. 3. Note that a small saturation threshold is used on Fig. 3b. It amplifies the difference in intensity of light and dark regions on an image and slightly improves the edge detection quality. Fig. 4 shows detection with weak constraints – it results in many false positive FIP detections, but the landmark still can be accurately detected.

There are several parameters that can be adjusted in the detection algorithm, but the exact set of parameters depends on the filtering that is applied to the input image:

- geometric constraints – the distance between FIPs and APs and their size ratio;
- color constraints – the color ratio used in FIP detection;
- filter parameters – for example, Canny's threshold, saturation threshold, blur kernel size, etc.

Given the location of FIPs and APs the perspective transformation can be computed in order to get orthogonal projection of the landmark and its position in the space. A QR code located in the center of the landmark can be extracted using that orthogonal projection and can be decoded using any QR code decoding algorithm (e.g. [3], [4]). It's clear that it still can't be done from far away, but the robot can always drive up to the landmark if it knows where the landmark is. An example of QR code extraction is shown in Fig. 5. Some postprocessing has been applied to the extracted QR code to get the image in Fig. 5b and most bar-code readers can decode the resulting QR, though the additional rectification may be applied.

IV. Landmark tracking

Landmark tracking is the next step for increasing detection robustness and quality. A robot can change the physical posi-

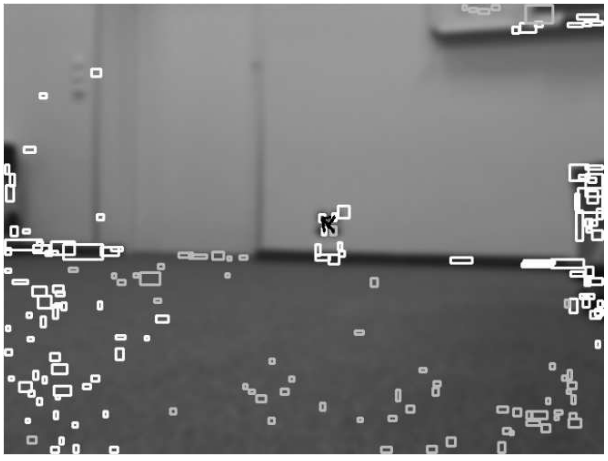
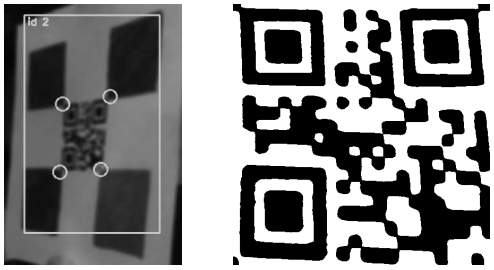


Fig. 4. Detection with weak constraints. The landmark in the center of the image still can be detected



(a) Detected label (b) Extracted QR code

Fig. 5. QR code extraction

tion of observation relatively smooth so the detected landmarks in the camera video stream will be relatively close on the two consecutive frames and therefore the detected landmarks may be tracked in a sequence of images. This can be done in the following way:

1) Initially there is an empty object (landmark) pool. Lifetime is assigned to every landmark and allows to keep the landmark in the pool for some time even if it isn't detected in several frames. The landmark is kept in the pool while its lifetime is less than the fixed maximum value.
2) Processing the next frame obtains a new set of landmarks. There are 3 different cases possible:
   a) If the object pool is empty then all new landmarks are stored in the pool and assigned ids.
   b) If the new set of landmarks is empty and the pool is not then the lifetime of all landmarks is increased and obsolete landmarks are removed.
   c) If both the pool and new landmark set are non-empty then it's required to solve the assignment problem where the pool landmarks are agents and the new landmarks are tasks (or vice versa), and the cost is the distance between a pool landmark and new one. So the total distance between old and new landmarks is minimized.
3) The obtained solution is checked to meet certain restrictions:
   - mapped landmarks are of similar sizes;
   - distance between them doesn't exceed the maximum value.

   If the mapped landmarks meet these restrictions then the position of the pool landmark is updated and its lifetime is reseted. Otherwise the lifetime of the pool landmark is increased and the new landmark is added to the pool.
4) All new landmarks that don't map to the pool landmarks are added to the pool.

Landmark tracking example is shown in Fig. 6. This algorithm is pretty standard and has 3 parameters that can be adjusted:

1) the landmark maximum lifetime depends on camera frame rate, but total time of 1 second looks appropriate;
2) the maximum distance between landmarks;
3) the landmark size ratio.

Specific settings of these parameters depend on robot physical features: maximum speed, camera frame rate, etc.

V. Evaluation

In this section the detection algorithm performance and robustness as well as some algorithm drawbacks are discussed.

A. Performance

The detection algorithm is supposed to be used on simple mobile robots with limited resources, so the testing environment based on popular low cost, credit-card sized computer Raspberry Pi [12] and Robotic Operation System (ROS) has been created.

ROS (Robot Operating System) [11] is a framework for robot software development that provides various system services such as hardware abstraction, low-level device control, implementation of commonly used functionality and message-passing between processes. Set of ROS processes is represented in a graph architecture where processing takes place in nodes that may receive and post sensor, control, planning and other messages. Simplified structure of the developed robot is shown in Fig. 7. The detector node works as a service that is polled by the main controller from time to time. Note that

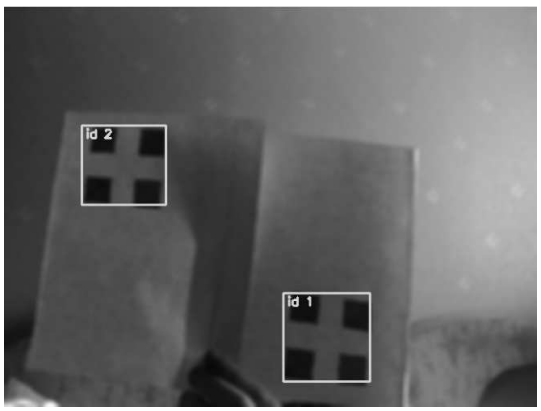


Fig. 6. Tracking example

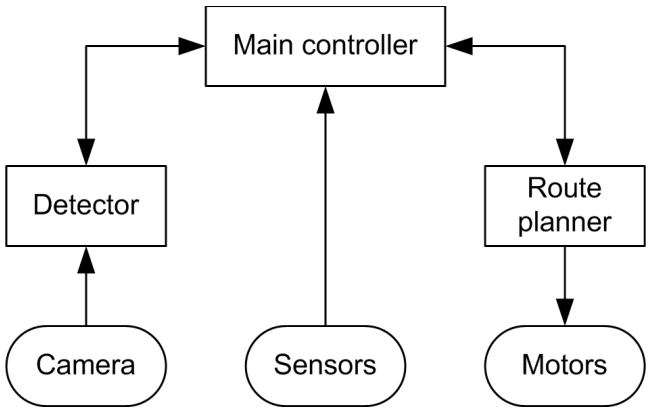


Fig. 7. Simplified robot structure

TABLE I
DETECTION ALGORITHM PERFORMANCE (FILTERS)

	640 × 480 (0.3 MP)				1280 × 720 (0.9 MP)				1920 × 1080 (2.1 MP)			
	Blur	HSV	Canny	Total	Blur	HSV	Canny	Total	Blur	HSV	Canny	Total
Intel Core i5	3.462	5.136	2.167	12.784	10.262	15.656	7.216	43.730	22.961	34.997	17.879	197.032
GeForce 650M	1.893	0.418	1.776	10.046 +2.258	4.845	0.798	4.058	23.855 +5.714	6.143	1.326	5.932	45.910 +11.976
HD Graphics 4000	2.153	0.501	1.874	25.661 +17.791	7.098	0.895	4.777	72.584 +55.751	10.876	1.643	8.480	191.327 +128.117
Raspberry Pi CPU	310.192	66.863	69.340	509.667	1123.2	191.653	213.537	1639.81	2737	441	493	3955
Raspberry Pi GPU	13.430	2.600	11.921	116.330 +8.744	28.901	6.093	29.974	258.378 +46.417	–	–	–	–

the ROS itself is not a real-time OS and message passing introduces some overhead.

In the initial implementation of the detection algorithm all image processing operations have been performed by OpenCV framework [10]. The performance of this algorithm implementation on the Raspberry Pi that utilizes 700 MHz CPU is around 1-2 FPS and when the detector is integrated in the whole robot-control system it handles only 0.5 frames per second. Profiling has shown that the most computationally expensive parts of the algorithm are denoising, HSV color space conversion and tresholding and edge detection. All these operations can be easily performed by GPU, so it was decided to transfer some parts of the algorithm to GPU.

There are several options for GPU computations:

- GPU-specific tools and technologies (e.g. nVidia CUDA) allow achieving high performance but only on the limited set of devices;
- OpenGL shaders are cross-platform but it's hard to perform non-image processing on shaders;
- using GPU assembler it's possible to achieve the maximum performance on the specific device but the development process is very effortful.

The second option has been chosen because it allows parallel development and testing on desktop computer and mobile robot. Currently only Gaussian blur, saturation extraction and Canny edge detector are ported to OpenGL. Algorithm testing has been performed on the following set of hardware:

- OpenCV implementation:
  - Intel Core i5 3210M, 2.5 GHz;
  - ARM1176JZF-S, 700 MHz (Raspberry Pi CPU);
- OpenGL implementation:
  - Intel HD Graphics 4000 (integrated);
  - nVidia GeForce 650M (discrete);
  - VideoCore IV (Raspberry Pi GPU).

Testing results for filter stages are shown in the Table I and for geometric stages – in the Table II. All timings are in milliseconds. In the Table I in column "Total" for GPU implementations the image transferring overhead is indicated.

The results show that for desktop systems the image transferring overhead is greater than the speed gain in filters especially for the integrated GPU, but as expected, on the big images GPUs are still more efficient than CPU.

On Raspberry Pi the usage of GPU gives 4.5x gain in overall performance that increase to 6x gain when increasing the image size. Unfortunately, the algorithm cannot process big images due to RPi limitations. Thereby the detector with GPU optimization can process 8-10 frames per second which is sufficient for simple mobile robots, but further optimizations may be investigated.

B. Robustness

To estimate detection rate we use two landmarks with 9 and 15 cm side and determine the maximum angle between landmark and camera planes at which 95% landmark detection rate can be reached. Results are shown in Fig. 8. Detector parameters have been tuned for every measurement so the graph shows the maximum reachable angle for reliable detection. In practice, static detector parameters adjusted to the environmental conditions allow to detect 9 cm landmark in $[-60°; 60°]$ horizontal angle range from 1.5 meters and 15 cm landmark in $[-75°; 75°]$ horizontal angle range from 2.5 meters.

The maximum distance at which landmark can be detected is determined by camera resolution. We used 0.3 MP webcam for testing and 9 cm landmark is indistinguishable in the image from approximately 2.5 meters and 15 cm landmark – from 4.5 meter. Thereby the landmark size is determined by the used camera and working environment features. In relatively small rooms with good lightning a 10-12 cm landmark is sufficient.

TABLE II
TOTAL DETECTION ALGORITHM PERFORMANCE (640X480)

	Filter	Geometry	Total	FPS
Intel Core i5	13.546	0.057	13.633	73
GeForce 650M	10.623	0.042	10.686	93
HD Graphics 4000	26.545	0.043	26.616	37
Raspberry Pi CPU	516.773	0.685	517.627	2
Raspberry Pi GPU	115.427	0.705	116.43	9

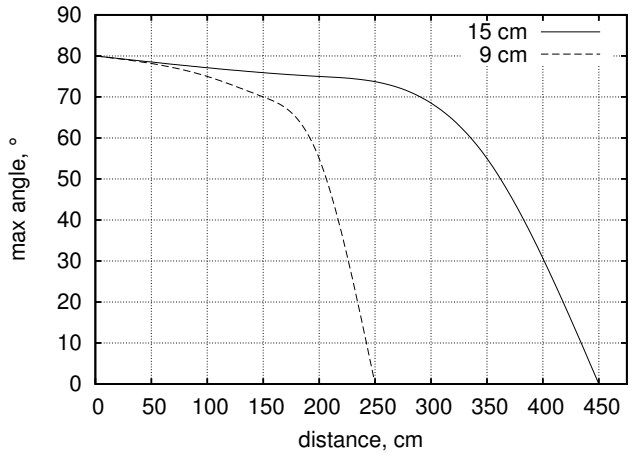


Fig. 8. Maximum angle for reliable detection

C. Drawbacks

The proposed landmark detection algorithm suffers from image noise like any other algorithm based on edge detection. Various denoising algorithms can be applied (Gaussian blur in our case), but there is a trade-off between overall performance and image quality. For example, large blur kernels can affect edge detection quality, but good denoising algorithms like non-local means [9] run extremely slow.

Another disadvantage of the algorithm is that it relies on color features that depend on external lighting. The landmark looks like light squares on a dark background in daylight and completely different in dim light. So detection algorithm parameters should be adjusted depending on the ambient conditions.

VI. Conclusion

The proposed landmark fits good landmark criteria: it is easy to create, set up, detect and identify. Although identification often requires approaching the landmark to read QR code, the fact of landmark(s) presence narrows amount of potential positions during localization process.

Described design and detection algorithm have various opportunities for enhancement:

- adaptive filters that can adjust their own parameters depending on environmental conditions;
- video stream for image stabilization, noise reduction and label tracking;
- experiments with colored FIPs and APs width can be performed to increase reliable angle range.

All programs and algorithm implementations are published as Open Sources Software and can be accessed by the following link: http://github.com/OSLL/landmark-detection.

Acknowledgment

Authors would like to thank JetBrains company for provided support and materials for working on this research. The paper has been prepared within the scope of project part of the state plan of the Board of Education of Russia (task # 2.136.2014/K).

References

[1] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots. *MIT Press*, 2011, p. 453.
[2] International Standard ISO/IEC 18004:2000 Information technology – Automatic identification and data capture techniques – Bar code symbology – QR Code, 2000.
[3] Luiz F. F. Belussi, Nina S. T. Hirata, "Fast Component-Based QR Code Detection in Arbitrarily Acquired Images", *Journal of Mathematical Imaging and Vision*, vol.45, no.3, Mar. 2013, pp. 277-292.
[4] Gabriel Klimek, Zoltan Vamossy, "QR Code Detection Using Parallel Lines", *Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium*, Nov. 2013, pp. 477-481.
[5] Yunhua Gu, Weixiang Zhang, "QR Code Recognition Based On Image Processing", *International Conference on Information Science and Technology*, Mar. 2011, pp. 734-736.
[6] Yue Liu, Mingjun Liu, "Automatic Recognition Algorithm of Quick Response Code Based on Embedded System", *In. proc. of the Sixth International Conference on Intelligent Systems Design and Applications*, Oct. 2006, pp. 783-788.
[7] Hao Wu, Guohui Tian, Peng Duan, Sen Sang, "The Design of a Novel Artificial Label for Robot Navigation", *Proceedings of 2013 Chineseintelligent Automation Conference*, pp. 479-486.
[8] Kuk-Jin Yoon, Gi-Jeong Jang, Sung-Ho Kim, In-So Kweon, "Fast Landmark Tracking and Localization Algorithm for the Mobile Robot Self-Localization", *IFAC Workshop on Mobile Robot Technology*, 2001, pp. 190-195.
[9] Antoni Buades, Bartomeu Coll, Jean-Michel Morel, "A non-local algorithm for image denoising", *In proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2005, pp. 60-65.
[10] OpenCV website, http://opencv.org
[11] ROS.org — Powering the world's robots, http://www.ros.org
[12] Raspberry Pi, http://www.raspberrypi.org