Experimental Platform and Matlab Toolbox for Planning Mobile Robots

Marius Kloetzer, Silvia Magdici and Adrian Burlacu

Abstract— **A test bed is developed for facilitating real time experiments with mobile robots. The setup consists of a wheeled mobile robot that moves on a platform, a computer that controls the robot and an overhead camera that closes the loop between these devices. Several Matlab routines were created for reducing the programming burden of designing an experiment. The environment in which the robot moves can be easily changed by placing colored convex regions (obstacles) on the platform. The interface between Matlab and the test bed is ensured by image processing functions that detect the environment map and the robot's position and orientation, and by functions for moving the robot. Thus, real experiments can be performed for various planning algorithms for which implementations are available. The functionality of the provided toolbox is enhanced by routines that plan the movement of a mobile robot based on cell decompositions, and a supporting experiment is included.**

I. INTRODUCTION

The application range for wheeled mobile robots increased rapidly over the last decades. Potential application can now include [1]: service robots for elderly persons, surveillance, automated guided vehicles for goods transfer, land mine detection or planet exploration. Any application involves perception of the environment, localization, map building, path planning and control.

Planning collision free paths for mobile robots is a significant research topic. The goal is to obtain a trajectory from a start position to a desired one while avoiding the objects that are part of the environment. Different algorithms were proposed to solve this type of navigation problems. The majority of these algorithms are usually tested and validated through simulation. A very challenging aspect emerges when their validation in real environments is needed. Multiple difficulties appear when measurements over approaches performance and experimental results comparison are needed.

Software tools allow users to develop or/and test, through simulation, navigation algorithms for mobile robots. Unfortunately, most are centered on having a reliable communication between host and robot, and lack flexibility when dealing with combined path planning algorithms [2], [3]. An important Matlab toolbox was developed by Corke [4] and allows the user to simulate real problems rather then trivial examples. Among the multitude of simulators for wheeled mobile robots reported in the literature, only a few are combined with an experimental platform. A soccer targeted platform is described in [5]. Programs are loaded and run on the robot, while the low battery autonomy robot must be charged repeatedly. In [6] an Internet-operated test bench (named *Teleworkbench*) for simulating robotic experiments is proposed. The architecture is complex, but the programs must be located in a place reachable by the Teleworkbench server, thus making it impossible for remote users to download code. Experimental results for multi-robot systems are reported in works as [7], [8], but the robotic platforms and the supporting software tools are dedicated to specific approaches.

M. Kloetzer, S. Magdici and A. Burlacu are with Faculty of Automatic Control and Computer Science, Automatic Control and Applied Informatics, Technical University of Iasi, D. Mangeron Street, No.27, 700050 , Romania. kmarius@tuiasi.ro; silvia.magdici@gmail.com; aburlacu@tuiasi.ro

Our work aims to design and build an experimental framework for mobile robots, which can be used to test multiple types of path planning algorithms developed by remote users. The framework is composed from an experimental platform and a navigation Matlab toolbox for wheeled mobile robots. In order to construct our test bed, both global information regarding the environment and mobile robot position are extracted through image processing from data provided by an eye in the sky type video camera. The representative path planning methods for static environments are: visibility graphs, Voronoi diagrams, grids, cell decompositions, potential fields [9], [10]. Each one has advantages and disadvantages, and for the current experiments we chose a cell decomposition method. This approach divides the free space of the environment into cells, represents the connectivity of the free space by the adjacency graph of these cells, and finds a collision-free path by a graph search.

The remainder of the paper is organized as follows. The design of the experimental platform is revealed in section 2, and the Matlab navigation toolbox is discussed in section 3. Experimental results are presented in section 4, conclusion and future works constitute the topic of section 5.

II. EXPERIMENTAL PLATFORM

We first outline the assumptions we make for all the experiments to be performed, and then we present the platform structure.

Assumptions:

We consider a mobile robot evolving in a rectangular-bounded environment where some regions of interest (*e.g.* obstacles) exist. The regions are disjoint, and any region has a convex polygonal shape.

The robot has a differential-wheel driven structure [10], *i.e.* the left and the right wheels can spin at different speeds, this leading to rotations in place, straight movements, or curved trajectories. The robot control inputs are the wheel speeds, which can be wirelessly sent to the robot from an external computer.

Moreover, the size of the robot is small with respect to environment bounds. As assumed in multiple planning meth-

ods, the robot position and orientation should be available for closing the control loop. Also, the environment map is needed, this consisting of positions of the existing regions.

Platform:

For accommodating the above assumptions, a Khepera III robot is used [11]. This robot is circular, with diameter of approximately 13 cm, height of 7 cm, and two traction wheels that are diametrically placed and can be independently controlled. It can communicate via Bluetooth with a computer. Among various commands, the wheel speeds are sent by the computer, and two PID controllers ensure that each wheel turns with the desired speed. The robot has multiple proximity and odometry sensors, and possible hardware extensions (as a board for autonomous driving, a gripper), but they are not of interest for this presentation.

For acquiring the robot position, we cover it with a white disc and we use a fixed overhead camera connected to the computer that controls the robot. The robot orientation is found by placing an off-center black dot on the white disc, towards robot's front side.

Since we want the robot to be small compared with the size of the environment, we use a high-definition webcamera with a wide viewing angle (namely Microsoft LifeCam Studio). The camera height was chosen such that the acquired image is clear enough, without distortions or color fading, and this yielded a platform size of almost 4 by 2.25 m. For diminishing light reflection, the platform has a black mate finish. Figure 1 shows the experimental test bed we built.

The regions from the environment can be easily made from colored paper or cardboard. White regions are not used for avoiding confusions with the robot, and currently red, green and blue colors can be correctly detected despite environmental light variations.

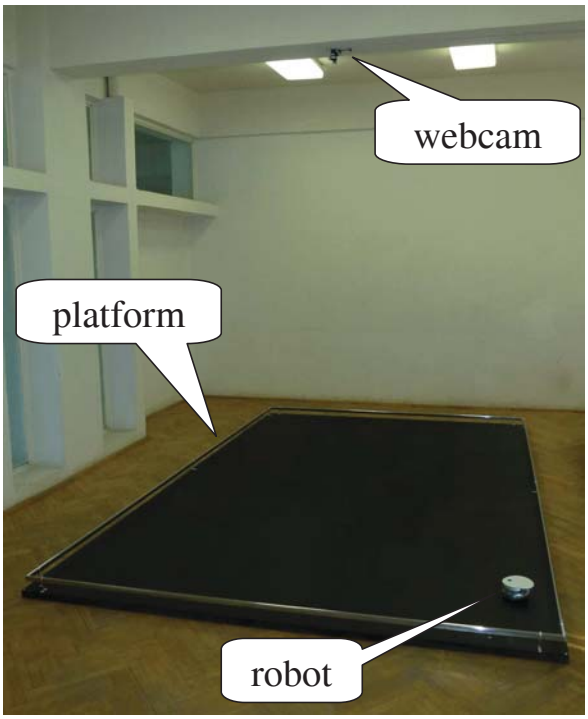

Fig. 1. Experimental test bed that consists of a black platform on which a Khepera III robot moves, and a surveying web camera.

III. MATLAB TOOLBOX

This section briefly presents the software routines we developed for allowing experimental tests on the described platform. These routines are grouped in a Matlab [12] package, and their purpose is to provide some general functionality such that different experimental tests with mobile robots can be constructed without a programming burden.

Subsection III-A describes the strategies used for recognizing the robot position and orientation, and for moving the robot towards a specific point (without accounting for any obstacle). Subsection III-B focuses on detecting the environment map (vertices of each region). After this interface with the platform is presented, section III-C adds some functionality that allows a move to target and obstacle avoidance experiment.

A. Robot position and control

Snapshots from the camera are captured by the computer, and by inspecting multiple sets of such images we defined RGB color component ranges for each color we handle. The small number of region colors yielded a calibration that is robust with respect to light variations, and thus each experiment can be performed without any prior calibration.

Once a snapshot is captured from the camera, the robot information is found by following a few simple steps:

(i) White regions are found based on the above mentioned color component ranges;
(ii) Possible perturbing regions with small area or non-circular are ignored, leading to a single remaining region;
(iii) The centroid of the detected region gives the position of robot's center in the environment reference frame;
(iv) The white region has a black hole, whose position is also returned for obtaining the robot orientation.

Based on the robot position, we develop a procedure for moving a specific point on the robot to a given position in the environment reference frame. There are two standard approaches allowing such a movement. The first one rotates the robot until its orientation is aligned with the direction towards the target point, and then it moves the robot on a straight line until its center is close enough to the target. Of course, a feedback strategy is needed for correcting deviations from the straight line. We use the second approach, which is inspired by [13], and it allows the off-center black dot from the robot to be regarded as a fully-actuated robot. In the following, the black dot is called the *reference* point, and its displacement from robot's center is denoted by ε. By denoting with u the desired bi-dimensional speed vector for the reference point (pointing from the reference point to the target one), by θ the robot's orientation, and by r the distance from its center to a wheel, the necessary speeds for the left and right wheels, v_L and v_R, are found from (1):

$$\begin{bmatrix} v_L \\ v_R \end{bmatrix} = \begin{bmatrix} 1 & -r \\ 1 & r \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & \varepsilon \end{bmatrix}^{-1} \cdot \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^{T} \cdot u, \tag{1}$$

Expression (1) comes from adapting for differential-wheel robots the input-output regulation problem solved in [13] for unicycle-type robots. By restricting magnitude of u such that $\|u\| \leq V_{max}\frac{\varepsilon}{\sqrt{\varepsilon^2+r^2}}$, the resulted wheel speeds are guaranteed to be included into a robot-specific interval $[-V_{max}, V_{max}]$.

Thus, for moving the reference point of the robot towards the target position, we iteratively capture an image, find robot's position and orientation and apply wheel speeds given by (1). The iterations are stopped when the reference point is close enough to the target one. The resulted motion is a straight line segment for the reference point, yielded by various trajectories for wheels. We mention that an advantage of this method over the method consisting in robot rotations and straight movements is its simplicity, since (1) includes a feedback control based on the actual robot position and orientation. As a disadvantage, the reduction of the robot to its fully-actuated reference point is done by increasing the obstacles slightly more than when the robot is reduced to its center (more explanations on this procedure are given in subsection III-C).

B. Environment map

1) Point Features: A point feature is a position that has a high image gradient in orthogonal directions. This condition may be fulfilled by single pixel that has a significantly different intensity to all of its neighbours or by a pixel on the corner of an object. Since corners are quite distinct they have a much higher likelihood of being reliably detected in an image sequence. Many algorithms have been proposed to detect point features [14]. For our application the Harris detector [15] was chosen. For this detector a point feature is considered being a location in the image where the local autocorrelation function has a distinct peak. Given an image $I(u,v)$, the autocorrelation matrix is computed from:

$$A = \sum_u \sum_v g(u,v) \begin{bmatrix} I_u^2 & I_u I_v \\ I_u I_v & I_v^2 \end{bmatrix} \tag{2}$$

where g is the Gaussian kernel and I_u, I_v are the gradients on the u and v directions. The main idea for detecting point features is to analyze the function:

$$C(A) = \det(A) - \delta \text{trace}^2(A), \tag{3}$$

where $\delta \in \mathbb{R}$ is a tunable sensitivity parameter. In order to establish the influence of δ let the two eigenvalues (which in this particular case are also the singular values) of A be $\sigma_1,\ \sigma_2$. Then:

$$C(A) = \sigma_1\sigma_2 - \delta(\sigma_1+\sigma_2)^2, \tag{4}$$

and for δ being small, both eigenvalues need to be big enough to make $C(A)$ pass a threshold, thus only point features are detected. Typically the corner strength is computed for every pixel resulting in a corner strength image. Then non-local maxima suppression is applied to retain only values that are greater than their immediate neighbours. A list of such points is created and sorted into descending corner strength. A threshold can be applied to only accept corners above a particular strength, or above a particular fraction of the strongest corner, or simply the strongest N corners. The algorithm proved to be robust due to its high reliability in finding L-junctions and its good temporal stability [15].

2) Region Detection: Information regarding the environment structure is obtained via a visual sensor. The region detection algorithm starts with the analysis of the image acquired by the overhead camera. Point features are extracted using the technique presented in III-B.1. The resulting set contains a large number of point features which must be associated with disjoint regions. These disjoint regions are obtained after each RGB components is segmented and a morphological operation over the resulting binary images is applied.

Once the point features are associated with disjoint regions, the assumption regarding the convex polygonal shape of the regions is employed. In order to increase the computation performance, a filtering stage is needed. This filter removes from each disjoint region the point features that do not modify the convex area of entire set more then a given threshold (5% was used in our experiments). The remaining point features are the ones that define the convex shape of each region.

C. Extended functionality

We intend to include in the developed toolbox multiple functions that can be useful when solving problems for mobile robots. Until now, routines that allow a motion planning problem via cell decomposition methods are available. The classical motion planning problem requires that a target position is reached by the robot's reference point, while obstacles (regions from environment) are avoided [9], [10]. Among different approaches proposed for automatically solving such problems, cell decompositions play an important role.

A cell decomposition is a partition of the free space (space not covered by obstacles) into convex regions having the same geometrical shape [16]. For planning the robot, a graph is constructed from the cell decomposition, where each node corresponds to a cell, and arcs correspond to adjacency relations between cells. The start cell is the one containing the initial position of the reference point, and the destination one contains the target position. A graph-search algorithm is employed for finding a sequence of cells to be traveled, and then the robot trajectory is found by linking the middle points of line segments shared by successive cells from the sequence. The initial and final position are linked at the beginning and the end of the obtained trajectory, and the result is a collision-free trajectory (composed from connected line segments) for the reference point. A simple example is given in Figure 2 for illustrating the main steps of this method when planning a point robot.

The control of the reference point on each line segment from the trajectory is done as in section III-A. Although the

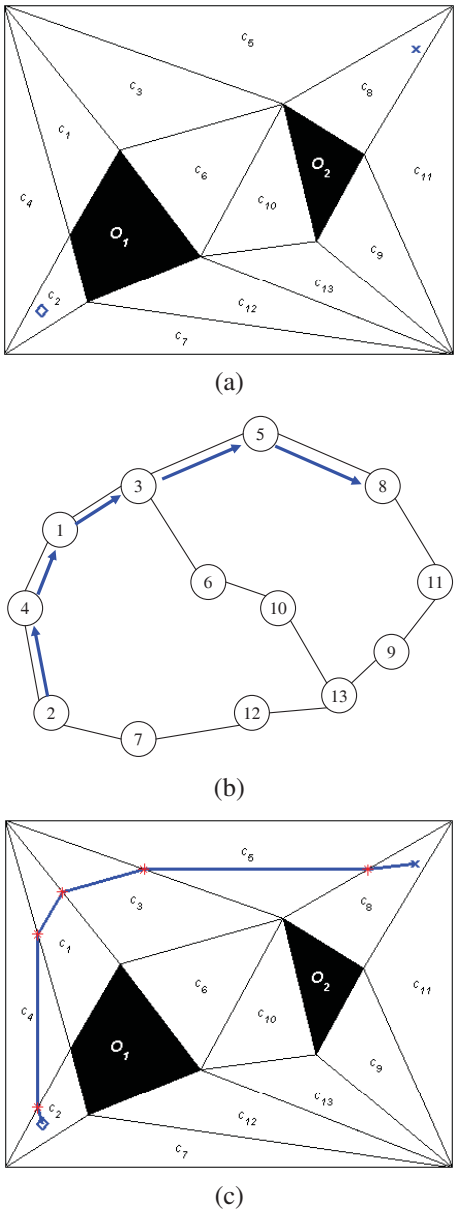

Fig. 2. Main steps for planning a fully-actuated point robot based on cell decompositions. (a) Environment with two obstacles, free space decomposed in triangular cells, start position marked with ♢, target position marked with X; (b) Corresponding adjacency graph and a path linking the start and the destination nodes; (c) The robot trajectory corresponding to the graph path (blue); the middle points of line segments shared by adjacent cells are marked with red.

reference point avoids obstacles, this is not yet guaranteed for the whole robot. For doing this, the robot must be reduced to its reference point prior to cell decomposition. This reduction is done by first over-approximating all possible rotations of the robot around its reference point with a convex polygon. Then, this convex polygon is slid along edges of every obstacle, and the positions spanned by the reference point define an increased obstacle. Similarly, the environment bounds are decreased, and the robot is reduced to its reference point. In our case, all robot rotations around its reference point yield a disc with radius $\varepsilon + r$, whereas when the robot movement is done by alternating rotations in place and straight movements, the disc radius is r. More details on reducing the robot to a point and on planning the motion by using cell decompositions can be found in [9], [10], [17], [18].

We include in our toolbox a method for increasing the obstacles such that the robot is reduced to a reference point, some Matlab cell decomposition routines reported in [19] and a Dijkstra graph search algorithm. Also, we include a method for imposing arc weights based on the expected traveled distance between adjacent cells. These functions can be easily combined for creating a motion planning experiment for the Khepera III robot, as illustrated in section IV. We intend to add more functions for planning robots based on potential functions, on Voronoi diagrams or on high-level motion specifications expressed in linear temporal logic formalism.

IV. Example

The facile usage of the developed test bed is illustrated in this section by conducting a motion planning experiment as explained in section III-C. The motion planning strategy is based on the following steps, whose implementation is included in the developed toolbox:

1) *Image acquisition*;
2) *Image preprocessing and point features detection*;
3) *Disjoint regions detection and point features filter*;
4) *Robot reduction to reference point*;
5) *Triangulation (cell decomposition) and trajectory planner*;
6) *Trajectory follow based on robot's position feedback*.

Figure 3(a) shows the initial deployment of the Khepera III robot in an environment cluttered with five obstacles (the red regions). The user selects with the mouse the target position for the reference point (marked with "X"). Figure 3(b) shows the outcome of the algorithm that recognizes the environment map (section III-B): the white robot is ignored, the background is removed, and the Harris corner detection yields the yellow points as vertices for every region. By removing the insignificant yellow points we end up with the corners marked with asterisks. In Figure 3(c) we depict the increased obstacles and shrunk environment bounds for reducing the robot to a point. These increased obstacles were obtained by over-approximating the robot rotation around its reference point with an eight-sided regular polygon.

In this environment, a fully-actuated point robot starting from the initial position marked with star should reach the final position marked with X. For finding a trajectory, in this example we use the cell decomposition method, briefly explained in section III-C. The triangular decomposition and the found trajectory are illustrated in Figure 3(d). The obtained trajectory is also represented in Figure 3(e), as the path that should be followed by the reference point. This trajectory is formed by line segments connected in the points marked with blue asterisks, and it is followed by the reference point by iteratively applying the procedure from section III-A. Some snapshots taken by the overhead camera

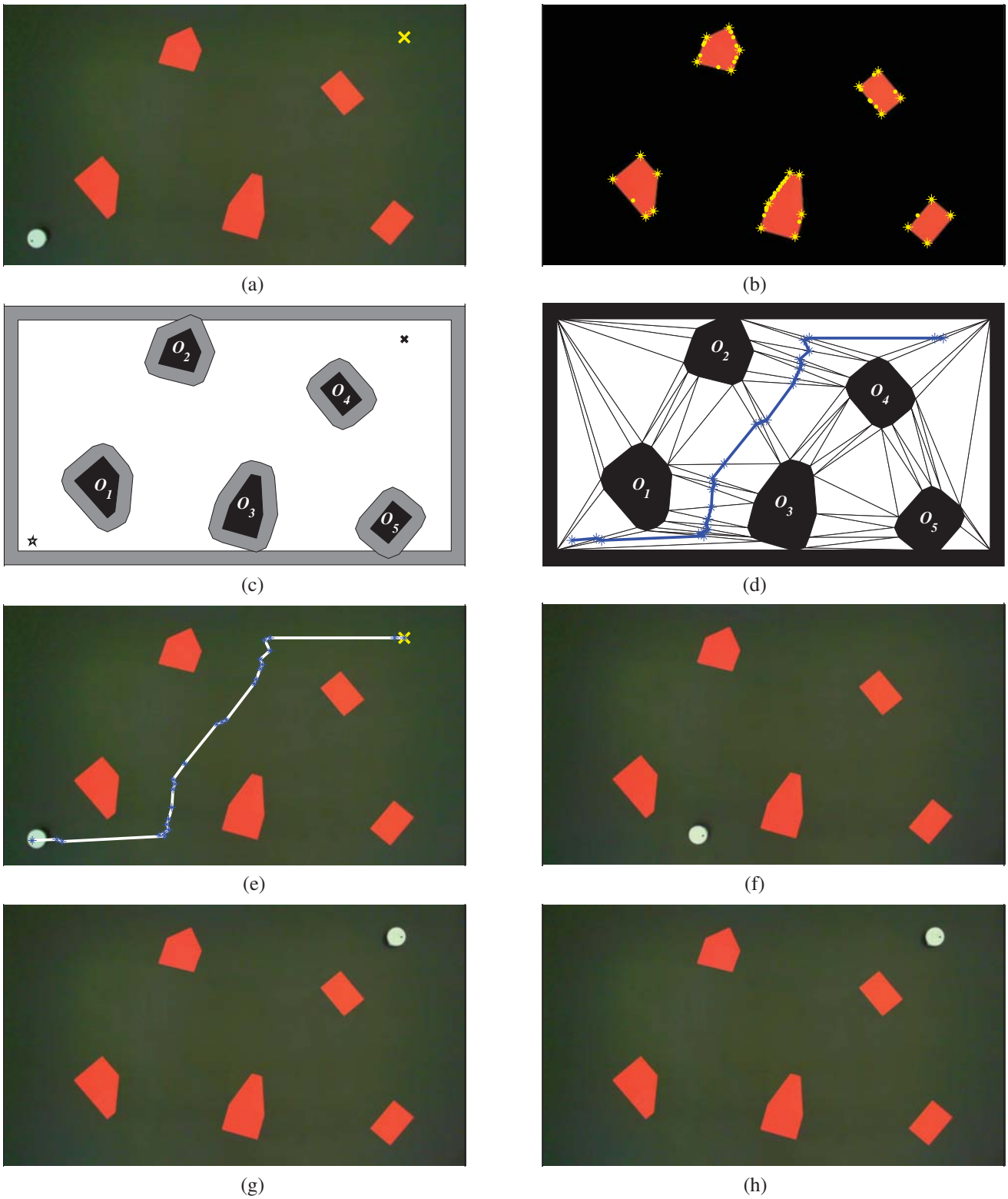

Fig. 3. Main steps of a motion planning experiment based on cell decompositions. (a) Environment with five obstacles (red), initial position of robot (white), and target position of the reference point (yellow X). (b) Corner points of each detected region are marked with yellow points, and from these the important vertices are emphasized with asterisks. (c) The grey area around obstacles and environment bounds mark their modification such that the robot is reduced to its reference point. (d) Triangular decomposition of the free space obtained after reducing the robot to a fully-actuated point, and the free-collision trajectory that should be followed. (e) Trajectory of the reference point overlapped on the initial environment snapshot. The trajectory is followed by iteratively moving the reference point towards the blue asterisks, as in section III-A. (f-h) Some platform images acquired by the overhead camera during the motion.

during the motion are given in figures 3(f-h), and a movie of this experiment can be found at [20].

A Matlab script that automatically plans the robot movement and controls it along the found trajectory is included in the developed toolbox, available for download at [20]. This script basically combines the routines explained in Section III, and the running time on a medium-performance laptop was less than 5 seconds until the trajectory is found. The robot speed was around 10 cm/s, based on the sampling time introduced by image acquisition and processing for finding robot's position.

V. CONCLUSION

The paper reports an experimental setup that can be used for testing and illustrating planning algorithms for which a feedback on robot's position and orientation is necessary. We employ a Khepera III robot that is wirelessly controlled by a computer, and an eye in the sky video camera that provides images needed for detecting the current robot position. The environment is cluttered with convex polygonal obstacles, and it can be easily defined by placing colored papers on the platform on which the robot moves. Matlab routines were developed for acquiring and simplifying the environment map, for finding robot's position and for moving it to a desired point. The provided toolbox includes functions that allow an experiment where the robot has to automatically reach a target point by avoiding obstacles. The movement is planned based on reducing the robot to a fully-actuated point and on finding a collision-free trajectory by using cell decompositions. A supporting example is presented, and future work will be conducted towards extending the toolbox applicability by including multiple routines that may be useful for planning mobile robots.

ACKNOWLEDGMENT

The first author acknowledges the support of the CNCS-UEFISCDI grant PN-II-RU code PD 333/2010. The third author acknowledges the support of the PERFORM-ERA project, ID - 57649, 2010-2013, POSDRU/89/1.5/S/57649.

REFERENCES

[1] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Springer, 2008.

[2] A. Elnagar and L. Lulu, "A visual tool for computer supported learning: The robot motion planning example," *Computers and Education*, vol. 49, pp. 269–283, 2007.

[3] J. Guzman, M. F. Rodriguez, and S. Dormido, "An interactive tool for mobile robot motion planning," *Robotics and Autonomous Systems*, vol. 56, pp. 396–409, 2008.

[4] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer, 2011.

[5] U. Witkowski, E. Monier, U. Ruckert, S. E. Ghoul, and M. El-Ghoniemy, "An automated platform for minirobots experiments," in *Proc. 10th Intl. Conf. on Control, Automation, Robotics and Vision*, Hanoi, Vietnam, 2008, pp. 685–688.

[6] F. Werner, U. Ruckert, A. Tanoto, and J. Welzel, "The telework-bench a platform for performing and comparing experiments in robot navigation," in *Proc. IEEE International Workshop on The Role of Experiments in Robotics Research (ICRA'2010)*, Anchorage, Alaska, USA, May 2010.

[7] G. Antonelli, F. Arrichiello, and S. Chiaverini, "Experiments of formation control with multirobot systems using the null-space-based behavioral control," *IEEE Transactions on Control Systems Technology*, vol. 17, pp. 1173–1182, 2009.

[8] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[9] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston: MIT Press, 2005.

[10] S. M. LaValle, *Planning Algorithms*, 2006, available at `http://planning.cs.uiuc.edu`.

[11] K-Team, "Khepera III robot," `http://www.k-team.com/mobile-robotics-products/khepera-iii`.

[12] The MathWorks, "MATLAB® 2010b," Natick, MA.

[13] J. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1998, pp. 2864–2869.

[14] C. Lazar and A. Burlacu, "Performance evaluation of point feature detectors for eye-in-hand visual servoing," in *Proc. IEEE Conference on Industrial Informatics (ICCP'2007)*, Cluj-Napoca, Romania, Aug. 2007.

[15] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry, *An Invitation to 3-D Vision*. Springer, 2003.

[16] M. D. Berg, O. Cheong, and M. van Kreveld, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, 2008.

[17] J. C. Latombe, *Robot Motion Planning*. Kluger Academic Pub., 1991.

[18] M. Kloetzer and C. Belta, "A framework for automatic deployment of robots in 2d and 3d environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[19] M. Kloetzer and N. Ghita, "Software tool for constructing cell decompositions," in *Proceedings of the IEEE Conference on Automation Science and Engineering*, 2011, pp. 507 – 512.

[20] "Experimental platform and Matlab toolbox for planning mobile robots," `http://www.ac.tuiasi.ro/~kmarius/RobPla/`.