Ballbot: A Low-Cost, Open-Source, Open-Design Robot for Tennis Ball Retrieval

John Wang, Karthik Lakshmanan, Ankush Gupta, Pieter Abbeel

Abstract— **In this paper we present an open robotic platform which is targeted to perform the task of a tennis ball boy. Our platform can localize itself and navigate on a tennis court. It can also localize tennis balls lying on the court. It is self-contained with on-board sensing and computation, uses only cost-effective off-the-shelf components, and was designed with an eye for robustness and repeatability in real-world environments. This paper describes our system, presents our first set of experimental results, and discusses some of the challenges faced.**

I. INTRODUCTION

Despite the success of the Roomba robotic vacuum, similar robotics platforms have been slow to follow. Roomba's success lay in its promise of accomplishing a simple repetitive task (vacuum cleaning) autonomously and at a competitive price [1].

We present a low-cost robot platform which, like the Roomba, is designed to perform a specific task (picking up balls during a tennis match) autonomously and reliably. A tennis "ball boy" robot must be aware of the environment around it. It needs to stay off the court during a match until it recognizes a dead ball, fetch the ball, and re-park itself at the edge of the court. It ought to do so with speed and precision.

In this paper, we present our first steps toward this goal and analyze some of the challenges involved. First we constructed a versatile mobile robot platform with a ball pickup attachment, and implemented a ROS software stack for base control and odometry. We then extended the capabilities of the software stack with a motion planner, a court localization module, and a stationary ball detector (as depicted in Fig. 1). Each component of the software stack is presented in detail, along with an analysis of experimental results.

The contributions of this paper are two-fold: (i) It provides an in-depth case study of the development of a robotic platform to be deployed by consumers. This study highlights use of state-of-the-art techniques in localization, navigation and control. (ii) It presents a low-cost, open-design, open-source robotic platform that we hope will enable other roboticists to build upon and has the potential to serve educational purposes. All our code is integrated with ROS [2], an open source codebase widely used in robotics.

A video illustrating our platform and its capabilities is available at http://rll.berkeley.edu/icra2012/ballbot.

II. PLATFORM

For this task, we built a mobile robot platform with on-board processing, computer vision, and wireless communication. We envision that the platform may be used to carry different payloads to accomplish different tasks. As the platform matures over the next couple of months, we will share the platform design so that the wider robotics community can benefit.

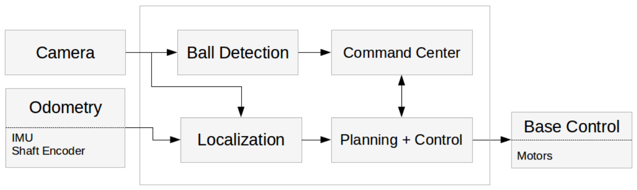

Fig. 1. System block diagram

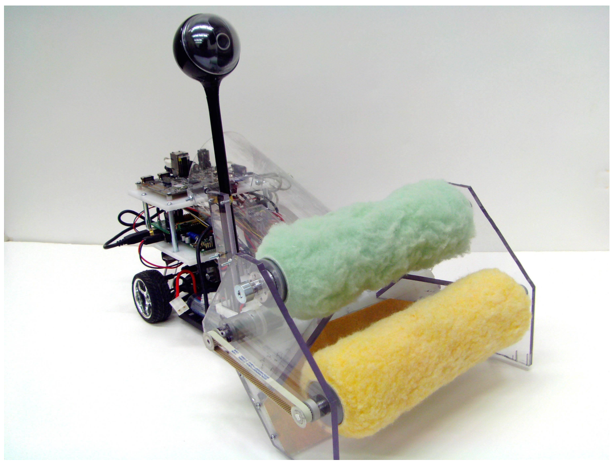

Fig. 2. The ballbot prototype with ball pickup attachment, side view

A. Mechanical

1) Base platform: The physical platform is built on top of a 1:10 scale RC car chassis, giving it good stability and handling at high speed. With a stiffened suspension, it can carry its weight (1.5 kg) in an additional end-effector and sensor payload.

2) Ball pickup: The ball pickup mechanism shown in Fig. 2 was designed to collect tennis balls from the ground, store the balls and deliver them back to the players. It uses two rollers which are independently driven through toothed belts by two DC motors. The rollers are placed horizontally parallel to the ground and are wide enough to tolerate an imprecise approach to the ball. The roll-out / roll-in can be achieved by reversing the direction of rotation of one of the rollers. This simple mechanism was found to work fairly reliably with only a 4% error rate.

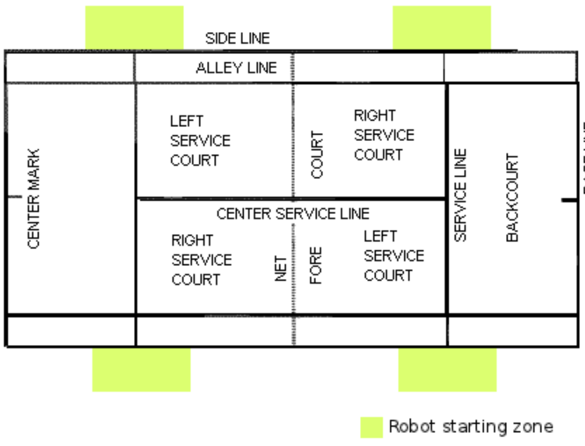

Fig. 3. Ideal initial placement zones for the robot

B. Control System

The main computing platform is the open-source PandaBoard[1], a dual-core 1GHz ARM Cortex-A9 computer which is roughly equivalent to a netbook in terms of computing power. The PandaBoard runs Ubuntu Linux, allowing the use of existing software frameworks like ROS and OpenCV, and regular webcams for vision. In general this platform allows for great flexibility in software and hardware choices.

Low-level motor and sensor interfacing is performed by an Arduino which communicates with the Pandaboard. Our sensor suite features an optical encoder and a 9DOF IMU. The Arduino is easily reprogrammable via USB and provides flexibility in interfacing different attachments or sensors.

C. Electrical

The entire platform is powered by a standard 7.2V NiMH battery pack. The robot is very power-efficient: a fully-charged battery can power the on-board computer and the motors for 3-4 hours during typical testing.

III. LOCALIZATION

There are three chief challenges to court line-based localization: lines are not unique, are relatively sparse, and convey less information than a point landmark. Some work has been done using court lines to supplement uniquely identified landmarks on the RoboCup field [3], and using constraint-based landmarks of varying types [4]. However, there are no unique landmarks on a tennis court, and lines on a court are not all visible to the robot at once, leading to ambiguity. Furthermore, tennis courts are highly symmetric, so a series of measurements while on the move are necessary to establish the robot position.

Because the inherent symmetry requires multiple hypotheses, we took a Monte Carlo Localization (MCL) [5] approach to localization using court lines as observed landmarks. To conserve computational resources on the embedded system, we constrain the problem further. First, we assume that the robot is initially placed by the sideline facing the service line, where it can see a corner (Fig. 3). This allows it to get a global initialization much more quickly. Second, to improve our accuracy, we constrain the robot's path to stay near court lines, similar to the coastal navigation presented in [6].

[1]More info available at pandaboard.org

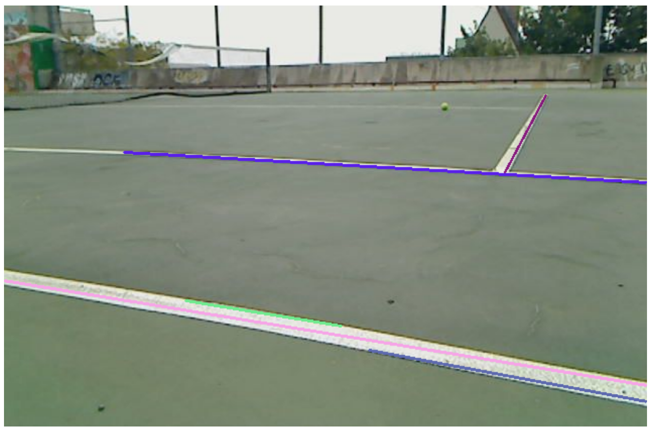

Fig. 4. View from the robot-mounted camera with detected lines labeled. Note the limited view at this height.

```
for p in particles:
  Generate the backprojection for court lines
   within 5m of particle
  w1, w2 = 1.0, 1.0
  for each detected line:
    w1 *= 1 + weight of most likely model line
  for each model line:
    w2 *= weight of most likely detected line
  p.weight *= w1 + w2
```

Fig. 5. Pseudocode for the MCL observation update

A. Line Detector

The line detector thresholds the image in grayscale to find the light-colored regions, then uses a probabilistic Hough transform [7], [8] to detect lines, as shown in Fig. 4. The Hough transform lines are then processed to remove duplicates using grouping by distance and similarity of angle. This line detector detects lines reliably within about 3 meters, but it also generates many false positives. The particle filter is able to handle these false positives.

B. Particle Filter

The MCL particle filter uses the detected lines as observations. Using the known tennis court model and the particle pose, it generates the backprojection of each court line onto the camera frame. Then, using a distance metric which takes into account both distance and orientation, the filter reweights the particles based on the observation according to the algorithm outlined in Fig. 5.

The observation update models the line detector's false positives by rewarding a matching line (small distance metric) but not penalizing for extra lines (large distance metric). However, expected lines which are not detected are penalized

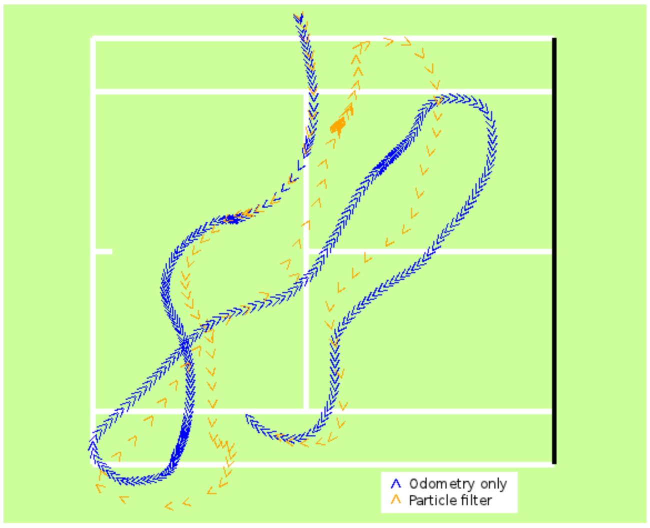

Fig. 6. Odometry vs. MCL estimated pose

slightly. This allows the lack of a line to inform the filter. The filter must be tuned not to penalize missing lines too much; otherwise particles that expect to see no lines, such as particles facing away from the court, may gain too much weight.

C. *Experimental Results*

Experiments were conducted on an outdoor tennis court in (1) midday, (2) sunset, and (3) overcast lighting conditions. The robot was steered by radio control. Odometry measurements and camera images were recorded for offline computation.

Initial results show that odometry alone looks qualitatively good but exhibits some drift. When court lines are in view, the particle filter corrects for drift. This correction can be seen in the yellow track in Fig. 6.

1) Backprojection model and noise rejection: Comparing detected lines and model lines in the image plane (using the backprojection model) was found to be much more robust to mechanical noise than comparing the lines in the world coordinate plane. This is because in the image plane, any mechanical vibrations which pitch the camera up and down will affect near and far lines equally.

2) Global vs. local localization: The particle filter can successfully perform a global localization from an arbitrary starting location. However, to perform the initial global localization, it was necessary to have about 5000 particles evenly spaced around the field. By relaxing this constraint and specifying that the ball must start in one of two starting locations, only about 200 particles are necessary to get an initial fix.

3) Running time: For 200 particles, each observation takes about 80ms on average using on-board processing. Therefore we are currently processing at about 12.5 frames per second. While this is sufficient, further code optimization should yield some performance gains.

IV. MOTION PLANNING AND CONTROL

The on-board motion planning and control framework is responsible for generating optimal and feasible paths in different scenarios, and for generating controls that move the robot from start to goal accurately. In essence, the planner is responsible for driving the car to a ball, retrieving the ball and delivering it to another location. The optimality of a plan is judged by its length, ease of control and time to compute given our limited computational resources and the need for quick response times during a tennis match.

A. *Path Planner*

The tennis court is a fairly simple environment from a planning point of view: it has fixed dimensions, is bounded on all four sides and has one consistent fixed obstacle—the net. A robot needs to account for other static obstacles such as benches and dynamic obstacles such as players. The planner is bound to respect both environmental constraints (obstacles) and differential constraints (e.g. a minimum turning radius of 0.7m for our robot).

The configuration of the robot is fully determined by a three dimensional vector (x, y, θ). Search based planning over a discretized configuration space is known to work well in such relatively low dimensional state spaces. In particular, lattice based planning [9], [10] is an approach that discretizes the configuration space into a set of states and connections that represent feasible paths between states. These connections can then be repeated to reach any configuration on the lattice. Lattice based planning effectively converts motion planning into a graph based search problem. It is well suited for our platform because it directly encodes differential constraints into the plan without the need for post processing. Moreover, it is easy to implement and compare various search algorithms without making large changes to the overall architecture of the planner.

Our planner is largely based on the framework provided by Pivtoraiko et al. [11], with a number of optimizations and simplifications to improve on-board performance. The different components of the planner will be explained below in accordance with the aforementioned structure.

1) Lattice state space and control set: The state lattice discretizes the tennis court into regular intervals of 0.175m, which is approximately 40% of the robot's length. For robot heading, we chose a minimal set of 8 uniformly spaced headings with an average out degree of 8. This was done in order to limit the branching factor and therefore reduce running time of the search. Motion primitives can be computed once from the origin and stored. The set of allowed motions from any lattice point can then be found by translating the stored motion primitives from the origin to that point.

2) Computing edge costs: Edge costs are typically computed by convolving the swath of the car with the cost map below [11]. We made two improvements that work well for our situation: (i) Convolving the swath with map cells for every edge in the graph search is expensive, but the sparsity of obstacles in our map allows us to heavily optimize by not convolving unless the edge is in close proximity to an obstacle. We obtained speed-ups of up to 50x in some cases, especially for paths around the net. (ii) We further penalize turns to generate straighter and more easily controllable paths.

3) Search algorithm: A* search [12] is a popular heuristic based search algorithm, which serves as our starting point. Our discretized search space, with low branching factor for each state, resulted in low run-times for A*. However, two issues require us to implement a better search algorithm with faster replanning: (i) The goal position may change during execution, either because the ball is moving or because the ball detector reports an updated location (ii) The robot might detect new obstacles that are not part of the map, like a player stepping in front of it.

In both cases, A* search in its original form will replan without using any information from previously generated paths. However, better search algorithms exist.

We use a version of Moving-Target (MT) search called Lazy MT-Adaptive A*, first introduced by Koenig et al [13] for the problem of quick path planning for characters in video games. Our results show that the algorithm works well for our situation as well, where both the agent (robot) and the goal (ball) can move.

MT-Adaptive A* is similar to A*, but is modified to incorporate two key ideas:

(i) Heuristic update after path computation:

For any state s that was expanded during an A* search, let $g(s)$ denote its g-value, i.e. the distance from the start state to state s. Let $g(s_{target})$ denote the g-value of the goal state s_{target}. Adaptive A* updates the h-values of all states s that were expanded during the search as follows:

$$h(s) := g(s_{target}) - g(s). \tag{1}$$

The new h-values are consistent and for any state, they cannot be smaller than the user-generated h-value for that state. Hence any new A* search using the new h-values will typically expand fewer states than the earlier searches.

(ii) Heuristic correction for Moving Target:

MT-Adaptive A* also corrects heuristics of nodes to maintain consistency when the goal changes. Given consistent h-values with respect to the previous goal state s_{target}, MT-Adaptive A* corrects the h-values of all states s to make them consistent with respect to the new goal state s'_{target}. It does this by assigning

$$h(s) := max(H(s, s_{target}), h(s)h(s_{target})) \tag{2}$$

for all s. It can be proved that the new h-values $h'(s)$ are consistent with respect to the new goal state s'_{target} [13]

MT-Adaptive A* with the new h-values cannot expand more states than an otherwise identical A* search with user-supplied initial h-values. In practice however, it usually expands much fewer nodes. The lazy version that we use does further optimizations to compute new h-values only for nodes that are needed during a future search. The entire algorithm is presented in detail in [13].

B. Controller

The robot has a closed loop controller that enables it to use localization information to follow planned paths accurately. The controller has two components—speed control and steering control.

1) Speed control: The controller commands speeds to the Arduino, which then runs PID control based on a wheel encoder to maintain the commanded speed. The controller uses a 0.25m lookahead to determine safe running speeds. This allows it to slow down before turns, when near obstacles and before reverse segments in the path.

2) Steering control: 1. Our steering controller is based on the one used by Stanley, Stanford's robot that won the DARPA Grand Challenge [14].

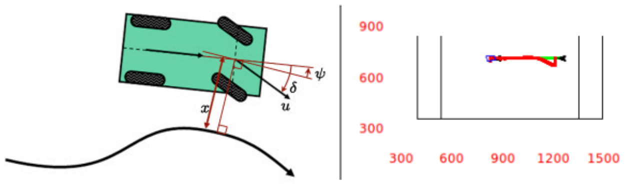

Fig. 7. a. Illustration of the Stanley steering controller [14] b. Stanley steering controller corrects trajectory from an inital cross-track error of 0.4m, k = 2 (Court markings are in cm)

The controller is based on a nonlinear feedback function of the cross-track error $x(t)$ which measures the lateral distance of the center of the robot's front wheels from the nearest point on the trajectory (Fig. 7). In the error-free case, using this term, the front wheels match the global orientation of the trajectory. The angle θ describes the orientation of the nearest path segment, measured relative to the vehicles own orientation. In the absence of any lateral errors, the controller points the front wheels parallel to the planner trajectory. $u(t)$ is the robot's speed at time t. The basic steering angle control law is given by

$$\delta(t) = \psi(t) + \arctan(\frac{kx(t)}{u(t)}), \tag{3}$$

where k is a gain parameter that can be tuned.

Using a linear bicycle model with infinite tire stiffness and tight steering limitations, it can be shown that for small cross track error, this control law results in error converging exponentially to zero [14].

C. Experiments

Fig. 8 shows various situations where the planner generates a plan and the controller drives the bot along the plan. All computation is done on board. These trajectories were recorded using the Vicon MX motion capture system, and superimposed onto a map of the tennis court. Table I displays quantitative measures of performance for both the planner and the controller, averaged over 10 runs per example. For the planner, the number of nodes expanded and runtimes provide a measure of performance, while the controller is measured by average cross-track error, final cross-track error and heading error at goal. Along with these average

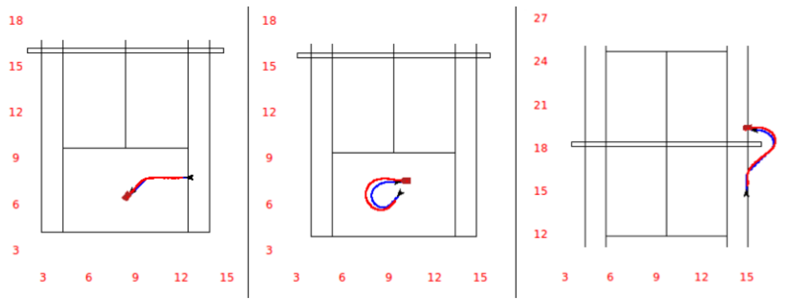

Fig. 8. Examples of plans generated (blue) and trajectories taken by car (red) for three cases — a) Picking up a ball b) Delivering a ball at a specific pose c) Retrieving a ball across the net. The court markings are all in meters

TABLE I

ON BOARD TESTS OF PLANNING AND CONTROL

Plan (from Fig. 8)	Nodes expanded	Run-time(s)	Average cross-track error(m)	Cross-track error at goal(m)	Heading error at goal(rad)
a)	104	0.48 (0.04)	0.05 (0.007)	0.067 (0.033)	0.1 (0.04)
b)	179	0.99 (0.47)	0.21 (0.025)	0.116 (0.0146)	0.13 (0.02)
c)	608	4.4 (0.42)	0.107 (0.007)	0.15 (0.017)	0.13 (0.09)

quantities, we also report the standard deviation as a measure of statistical significance of our results. We can see that although there is room for improvement with the planner's speed, it does a satisfactory job of generating initial plans. The controller performs very well. As an additional measure of the controller's performance, we can report a 93% success rate for the robot arriving at the goal such that the ball is encased within its roller.

V. BALL DETECTION

A. Approach

We developed a novel approach for locating stationary balls on the surface of a tennis court. In order to cut out impossible regions where stationary balls could be found and reduce the search domain, only the region below a finite horizon extending over the court length is considered. The approach assumes that at most one ball is present in the frame. It further assumes that fine texture details in the image are redundant. The textures and noise are smoothed by running mean shift like clustering over space [15]. The resulting posterized image has larger areas with nearly constant colors. Connected components or blobs, which are contiguous regions of nearly constant color are generated from the posterized image using a flood fill algorithm. Connected components which have pixel area much greater than what is possible for an image of a taken ball from the camera's height are discarded.

Contours bounding the connected components are found [16]. Contours help in obtaining useful geometric properties like area, shape and position of the blobs. Contours are filtered on size and shape. This filtering based shape is done by first fitting ellipses to the blobs using a least squares technique [17] and then evauating the following two measures,

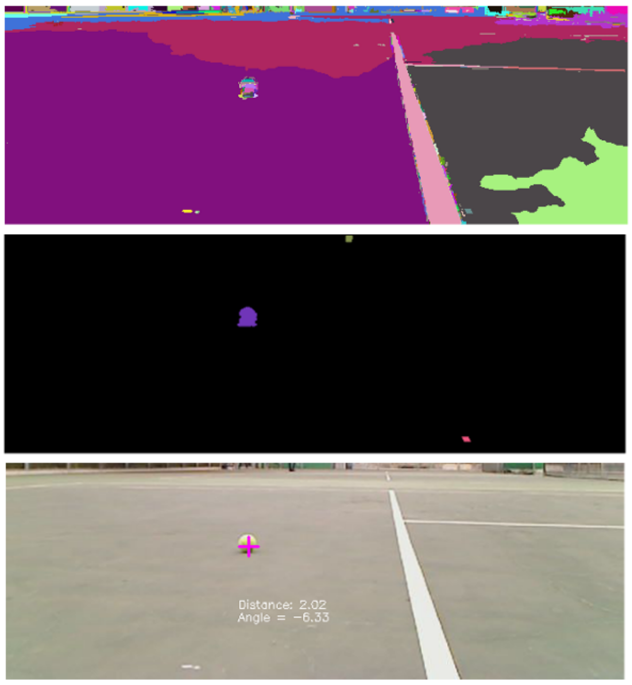

Fig. 9. Intermediate steps of ball detection. *[from top]* a. Connected components after mean-shift clustering and flood-filling b. Blobs left after filtering on shape, size and color. c. Detected ball and estimated position (meters, degrees)

$$\rho := M/m \tag{4}$$

where M = length of major axis, m = length of minor axis,

$$\Delta := 1 - \frac{Area(Blob)}{Area(Ellipse)} \tag{5}$$

For a circular blob, it is expected that $\rho \to 1^+$ and

$\Delta \to 0^+$.

The remaining contours' pixels are then converted to HSV colorspace for color-based filtering aided by the tennis ball's distinct color. In the rare case that multiple blobs remain, the one with the largest area is assumed to represent the ball (see Fig. 9). Successive averaging and filtering leads to a progressively diminishing set of ball candidates which aids in reducing the computation overhead, a precious resource for embedded systems.

B. Experiments

This approach was found to detect balls much farther away (5-6 meters) than naive color thresholding which was only useful for balls within close range (1-2 meters). It was more robust to potential detractors like distant trees, round objects like stones or even varying lighting conditions. It was found to have a pleasingly low false negative rate but a relatively higher false positive rate, which was primarily due to random misdetection and can be improved by imposing constraints on position of the ball based on the motion feedback of the robot.

VI. CONCLUSIONS AND FUTURE WORK

Our efforts to develop a low-cost integrated system for tennis ball retrieval have thus far resulted in the individual capabilities of localization, navigation and control, and ball detection. We are currently working on the integration of these components. In the further future it will require the ability to sufficiently understand the status of the game to know when to retrieve a ball and where to deliver it.

We have come a long way in fulfilling a secondary goal, which is to develop a versatile low-cost research platform that can handle a high top speed and accommodate different end-effectors or sensor payloads. We believe this open platform will benefit other researchers and robotics hobbyists.

REFERENCES

[1] J. Jones, "Robots at the tipping point: the road to irobot roomba," *Robotics Automation Magazine, IEEE*, vol. 13, no. 1, pp. 76 – 78, march 2006.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," 2009.

[3] T. Rofer, T. Laue, and D. Thomas, "Particle-filter-based self-localization using landmarks and directed lines," in *RoboCup 2005: Robot Soccer World Cup IX*, ser. Lecture Notes in Computer Science, A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, Eds. Springer Berlin / Heidelberg, 2006, vol. 4020, pp. 608–615, 10.1007/11780519_60. [Online]. Available: http://dx.doi.org/10.1007/11780519_60

[4] A. Stroupe, K. Sikorski, and T. Balch, "Constraint-based landmark localization," in *RoboCup 2002: Robot Soccer World Cup VI*, ser. Lecture Notes in Computer Science, G. Kaminka, P. Lima, and R. Rojas, Eds. Springer Berlin / Heidelberg, 2003, vol. 2752, pp. 8–24, 10.1007/978-3-540-45135-8_2. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45135-8_2

[5] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99 – 141, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370201000698

[6] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal navigation-mobile robot navigation with uncertainty in dynamic environments," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1, 1999, pp. 35 –40 vol.1.

[7] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, pp. 11–15, January 1972. [Online]. Available: http://doi.acm.org/10.1145/361237.361242

[8] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119 – 137, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314299908317

[9] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009. [Online]. Available: http://ijr.sagepub.com/content/28/8/933.abstract

[10] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, aug. 2005, pp. 3231 – 3237.

[11] M. Pivtoraiko, R. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, March 2009.

[12] N. J. Nilsson, *Principles of Artificial Intelligence*, Nilsson, N. J., Ed., 1982.

[13] S. Koenig, M. Likhachev, and X. Sun, "Speeding up moving-target search," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, ser. AAMAS '07. New York, NY, USA: ACM, 2007, pp. 188:1–188:8. [Online]. Available: http://doi.acm.org/10.1145/1329125.1329353

[14] S. Thrun et al, "Stanley: The robot that won the darpa grand challenge," in *The 2005 DARPA Grand Challenge*, ser. Springer Tracts in Advanced Robotics, M. Buehler, K. Iagnemma, and S. Singh, Eds. Springer Berlin / Heidelberg, 2007, vol. 36, pp. 1–43, 10.1007/978-3-540-73429-1_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73429-1_1

[15] D. Comaniciu and P. Meer, "Mean shift analysis and applications," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1197 –1203 vol.2.

[16] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32 – 46, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0734189X85900167

[17] M. Pilu, A. Fitzgibbon, and R. Fisher, "Ellipse-specific direct least-square fitting," in *Image Processing, 1996. Proceedings., International Conference on*, vol. 3, sep 1996, pp. 599 –602 vol.3.