Ballbot: A Low-Cost Robot for Tennis Ball Retrieval

John Wang

June 2, 2012

Acknowledgement

Pieter Abbeel, Karthik Lakshmanan, Ankush Gupta

Ballbot: A Low-Cost Robot for Tennis Ball Retrieval

John Wang

Abstract— **This paper presents a robotic platform which is targeted to perform the task of a tennis ball boy. This platform is self-contained with on-board sensing and computation, uses only cost-effective off-the-shelf components, and was designed to perform robustly and repeatably in semistructured real-world environments. In particular, this paper presents a particle-filter implementation that allows the robot to localize itself and navigate on a tennis court. Along with navigation and tennis ball detection, this will enable it to pick up tennis balls on a real tennis court. This paper describes our system, presents our initial 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 more aware of the environment than a Roomba. Rather than wandering about blindly, it needs to be able to find its own location and navigate to different points of interest as the situation warrants. Specifically, 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.

In this paper, we present our first steps toward this goal and analyze some of the challenges involved. First we constructed a 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 hardware and software systems are described in this paper, along with an analysis of problems encountered in their implementation.

The emphases of this project are two-fold. First, it provides an in-depth case study of the development of a robotic platform to perform a specific task autonomously, highlighting the use of state-of-the-art techniques in localization, navigation and control. Second, it presents a low-cost robotic platform that has the potential to serve wider research and educational purposes.

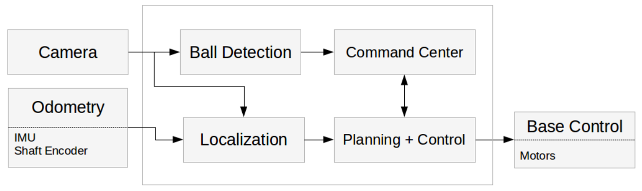

Fig. 1. System block diagram

II. HARDWARE PLATFORM

For this task, we built a mobile robot platform with on-board processing, computer vision, and wireless communication. The platform is designed to be self-contained and perform all computations on board. We envision that the platform may be used to carry different payloads to accomplish different tasks. Its relatively low cost and availability of its constituent parts makes it attractive for educational use, particularly at the high school and university level.

A. Mechanical

1) Base chassis: The robot is built on top of a 1:10 scale RC car chassis. We decided to use an off-the-shelf chassis for increased reliability. The Ackermann steering geometry of the car lends it good stability and handling at high speed. The car is a scale model sedan meant for hobby racing; therefore, the chassis rides very close to the ground. The tennis court is a flat, regular surface which the on-road tires are well suited for.

The entire robot weighs 4 kg, where the stock car base is 1.5 kg (including the battery) and the payload is 2.5 kg. This added weight required stiffening the suspension using higher rate springs and adding preload to the springs. (The high-rate springs we used are 700 N/m, so a compression of approximately 1.3cm on

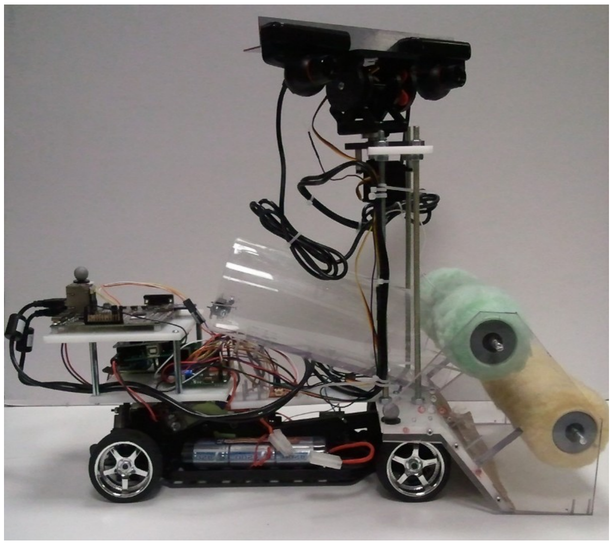
Fig. 2. The ballbot prototype with ball pickup attachment, side view

each suspension was necessary to support the car's weight.) Most of the robot's payload weight comes from the thick sheets of HDPE and polycarbonate used to make the end-effector and electronics mounting plates. Judicious use of thinner plastics or alternate materials could have reduced the weight substantially.

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 horizontally positioned rollers which are independently driven through timing belts. (The use of timing belts enables more accurate speed control, for example to launch the ball at varying speeds; however, this was never necessary in practice.) The motors chosen to power the pickup were 25mm-diameter high-power motors with 75:1 gearboxes sold by Pololu. The motors draw 6A at stall, providing 130 oz-in of torque, which was a ballpark figure we would need to compress the tennis ball as it enters each roller.

The opening of the rollers is 24 cm wide, which is wide enough to tolerate an imprecise approach to the ball. The balls are stored in a downward-sloped tube which accommodates up to 3 balls. Ejecting balls is achieved by reversing the rotation of the top roller, launching the ball through the middle of the two rollers. This simple mechanism was found to work fairly reliably with a 96% success rate in autonomous pickup tests.

B. Power Electronics

1) Main Drive Motor: Control of the main drive motor is provided by an off-the-shelf Pololu Simple 18v25 motor controller, which can handle up to 25A continuous current. This is enough to drive the main drive motor, which is a stock Tamiya (Mabuchi) RS-540 motor that stalls at around 34A at 7.2V. The decision to buy a motor controller rather than build one was motivated by the difficulty of building and debugging a discrete H-bridge motor driver that could handle the necessary current.

2) Pickup Motors: Control of the two smaller ball pickup motors is provided by a custom relay driver board, whose relay contacts can handle 8A. The driver board uses two DPDT relays, allowing one motor (the top roller) to be reversed. This board only handles on/off motor switching without variable speed control. The decision to build a relay board was motivated by the simplicity of bidirectional control (vs. an H-bridge) and the need for only on/off control. Off-the-shelf motor control solutions would also have worked, but were not pursued at the time.

3) Power Supply: The Zeus 3A from Basic Micro, a buck-type switching regulator, supplies the Pandaboard with 5V from the 7.2V battery. This works somewhat, but the lack of a boost regulator (and the large currents drawn by the drive motor) makes the Pandaboard susceptible to brown out when the battery is at about 75% charge. This should be addressed, but has been temporarily mitigated by keeping the batteries fresh and occasionally using a second battery to power the Pandaboard.

C. Control Electronics

Our platform has a two-tiered control system, where compute-intensive, non-realtime tasks such as particle filtering and vision processing are performed on a separate microprocessor from real-time tasks such as actuator and sensor interfacing.

1) PandaBoard: The PandaBoard is used as the main processor and is responsible for compute-intensive tasks and external communications. It uses the OMAP4430, a dual-core 1GHz ARM Cortex-A9 processor, which is roughly equivalent to a netbook in terms of computing power. It features an onboard WiFi module, Ethernet, and 2 USB ports.

2) Arduino Mega: The Arduino Mega is used for low-level motor and sensor interfacing. It is an ATmega2560-based development board which is clocked at 16MHz and features 8kB SRAM, 4 hardware

UARTs, a 2-wire interface for I2C, and 16 ADC channels. This board communicates with the Pandaboard via a hardware serial port on the OMAP processor. The hardware serial is important because USB-serial incurs a non-deterministic latency from the USB-serial driver stack in a non-realtime operating system, which may result in random localization errors and destabilize the control loop.

D. Sensors

The Arduino Mega board is easily reprogrammed via USB and provides flexibility in interfacing different attachments or sensors. Our sensor suite features a quadrature optical encoder and a 9DOF IMU.

1) Wheel Encoder: The optical encoder is used both for velocity control and for odometry. It uses an Omron EE-SX1031 slot-type dual photointerruptor and an appropriately-sized codewheel to provide a resolution of 204 counts/m.

2) IMU: The IMU module used is the commercially available SparkFun 9DOF IMU. It provides stable, filtered yaw rate measurements for odometry. The IMU is mounted on the centerline of the car for accurate yaw measurements. It is not drift-free, but it has proven reliable enough for incremental measurements of odometry.

3) Stereo Vision: Two Playstation Eye cameras are interfaced with the PandaBoard via USB. The Playstation Eyes have many features that make them suitable for computer vision on a budget. They provide a high framerate of 100 frames/s at 320x240 resolution, have a relatively low-latency Linux driver which bypasses the weakness of traditional UVC cameras, and are commercially available for under $30. The two cameras are mounted in a parallel stereo configuration on a rigid piece of aluminum bracket. The camera pair is then mounted on a pan/tilt servo mount which allows full freedom to pan and tilt +/- 90 degrees. The camera centers are aligned with the tilt axis so that the camera height remains constant as the camera mount tilts.

A stereo camera calibration was performed on the mount to find both the individual camera parameters and the stereo camera parameters. In this process the distance between cameras was found to be 16.2 cm.

E. Power Source

The entire platform is powered by a standard 7.2V NiMH battery pack. The robot has fairly good endurance. A fully-charged battery can power the onboard computer for 3-4 hours during typical testing, although actual battery life will depend on how often the drive motors are run.

F. Robustness

Hardware robustness contributes greatly to software robustness. Conversely, brittle hardware can affect the performance of otherwise-robust algorithms. This was experienced many times throughout our development cycle as hardware problems came up. The latency problems encountered with USB-serial communications are an example of how an overlooked hardware problem caused a difficult-to-diagnose software problem.

Another such occurrence happened with the original camera, a Logitech QuickCam Orbit. It was selected for its integrated pan/tilt functionality. However, after a few months of operation, the internal pan/tilt mount became loose. Our vision data became corrupted with large amounts of up/down vibration. The solution was to buy a real pan/tilt camera mount and switch to a different camera that could be affixed on the mount.

III. SOFTWARE PLATFORM

The PandaBoard runs Ubuntu, a Linux-based OS. Using a widely-available open source operating system ensures compatibility with existing software frameworks such as ROS and OpenCV. It also enables the use of commodity hardware, such as USB webcams for vision. The Linux environment provides rich APIs and abstracts the software away from the choice of hardware (i.e. the ARM-based OMAP4). In principle, it would be simple to migrate to an Intel-based processor should the need arise. The use of a Linux-based OS allows for great flexibility in software and hardware choices.

A. Arduino Mega Software

The Arduino Mega is programmed with software necessary to interface with the motors, servos and sensors. It runs a message-processing loop to receive commands (velocity, steering angle, ball pickup state) from the PandaBoard.

The Arduino Mega runs a few time-critical tasks. Foremost of these, it runs a PID control loop to set the speed of the robot. This is done on board the low-level microcontroller because the control loop requires precise timing stability. It also sends odometry readings back to the PandaBoard at regular intervals. This ensures that the readings are spaced evenly so that the time delta between readings is consistent.

The Arduino Mega only needs to be programmed once with the base controller software. It has many more hardware peripherals, GPIO pins and ADC inputs allowing the robot to support flexible hardware configurations.

B. PandaBoard Software

PandaBoard software is developed onboard through an SSH connection. The SSH connection is also used to run and debug software. All our code is integrated with ROS [2], an open source codebase widely used in robotics. ROS provides a publisher / subscriber model for different nodes, or processes, to communicate. ROS handles communications between ROS nodes running either on the same computer, or on a different network-connected computer. ROS also provides `rosbag`, a tool to record and playback timestamped messages for offline computations.

IV. LOCALIZATION

In order to determine our position on court, we used the computer vision system to track court lines, and used visible lines as well as motion cues to deduce our position.

The chief challenge faced in using court lines as features is that court lines are relatively sparse and non-unique, and they only constrain the robot's position in 1 dimension. That is to say, from observing a court line, the robot can determine its heading and distance from the line, but not its lateral position along the line. The posterior distribution from that one observation, then, is two parallel lines along the court line.

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].

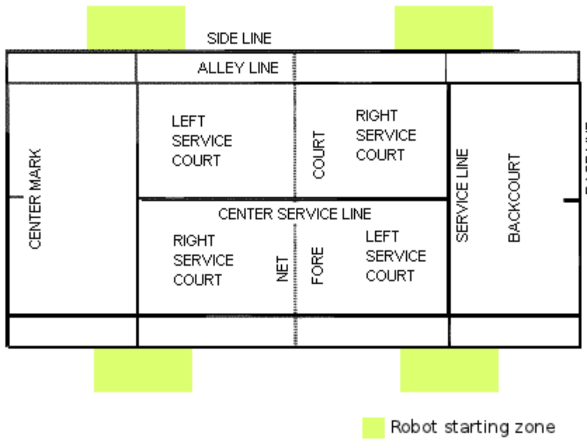

Fig. 3. Ideal initial placement zones for the robot

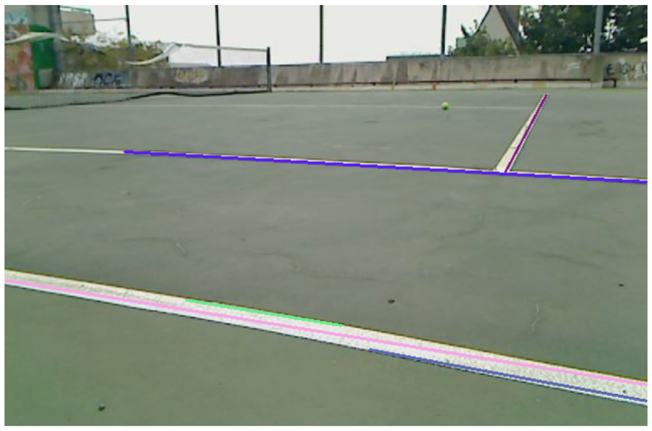

Fig. 4. View from the robot-mounted camera with detected lines labeled. Note the limited view at this height.

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

```
for each particle $(x_t^{(i)}, w_t^{(i)})$ do
    $w_1 \leftarrow 1.0$
    $w_2 \leftarrow 1.0$
    for each observed line $z_t^{(j)}$ do
        $w_1 \leftarrow w_1 * (1 + p(z_t^{(j)} | x_t^{(i)}))$
    end for
    for each expected line $\hat{z}_t^{(k)}$ do
        $w_2 \leftarrow w_2 * \max_j p(z_t^{(j)} = \hat{z}_t^{(k)} | x_t^{(i)})$
    end for
    $w_t^{(i)} \leftarrow w_t^{(i)} * (w_1 + w_2)$
end for
```

Fig. 5. Pseudocode for the MCL observation update

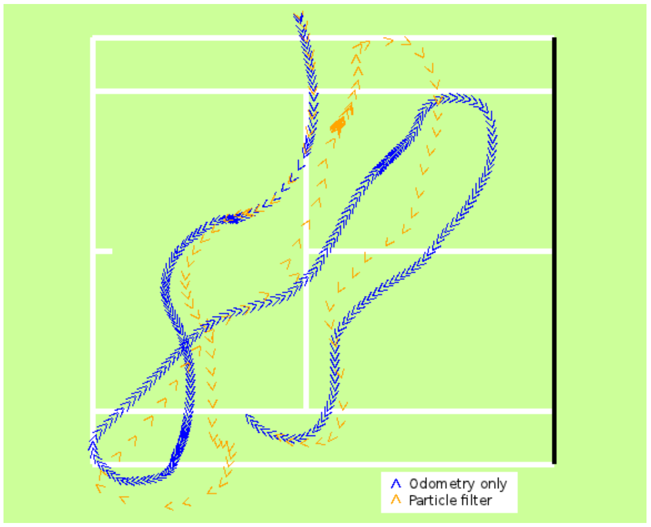

Fig. 6. Odometry vs. MCL estimated pose

and orientation, the filter re-weights 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 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.

D. Next Steps

Vanilla MCL localization is currently sensitive to global initialization failures. A major feature of tennis courts is the symmetry. A bimodal particle distribution can easily devolve into a unimodal distribution when there is not enough evidence to differentiate between two hypotheses, even using a low-variance sampler. One simple solution, which must be tested more thoroughly, is to resample the particles less frequently. Another solution which has been pursued to a limited degree is to inject random particles if the weight sum falls below some threshold. A more sophisticated solution may involve using a Mixture MCL as described in [5]. This particle filter sometimes uses the observation as the proposal distribution, drawing likely poses from the current observation. It may be more a more intelligent and principled way to overcome global initialization failures than the random particle injection method.

V. 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

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

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].

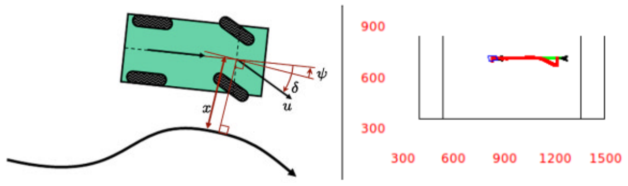

Fig. 7. a. Illustration of the Stanley steering controller [14] b. Stanley steering controller corrects trajectory from an inital cross-track error of 0.4m, k = 2 (Court markings are in cm)

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 vehicle's 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

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

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 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.

VI. 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

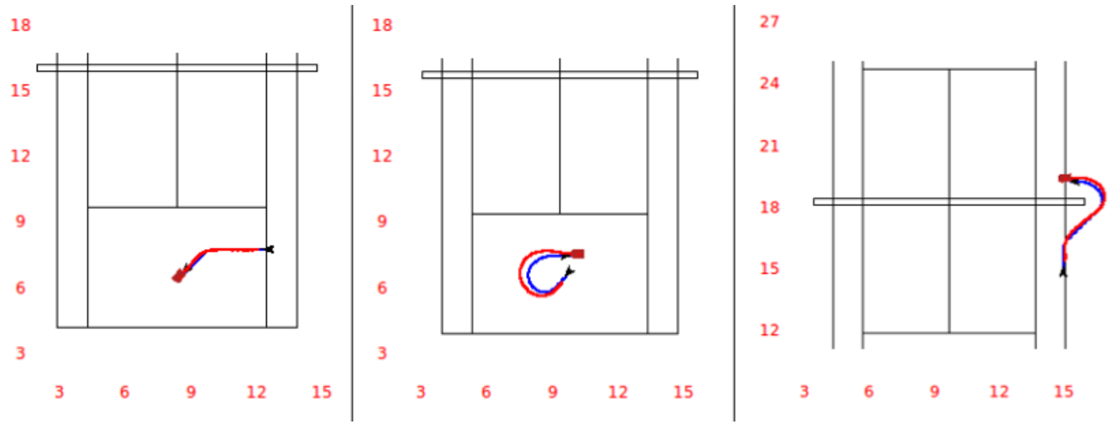

Fig. 8. Examples of plans generated (blue) and trajectories taken by car (red) for three cases — a) Picking up a ball b) Delivering a ball at a specific pose c) Retrieving a ball across the net. The court markings are all in meters

TABLE I
MEAN ERROR (S.D.) FROM ON-BOARD TESTS OF PLANNING AND CONTROL

Plan (from Fig. 8)	Nodes expanded	Runtime (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)

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,

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

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.

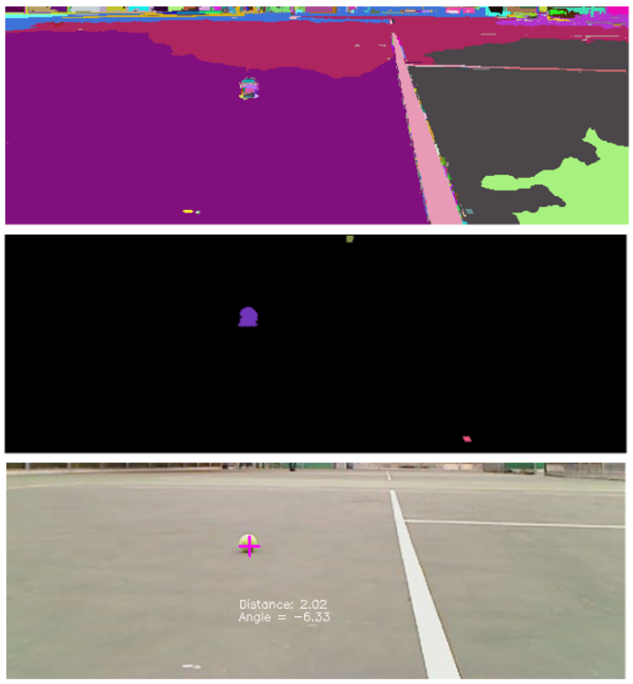

Fig. 9. Intermediate steps of ball detection. *[from top]* a. Connected components after mean-shift clustering and flood-filling b. Blobs left after filtering on shape, size and color. c. Detected ball and estimated position (meters, degrees)

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 low false negative rate. It tends to have a higher false positive rate than color thresholding, which was primarily due to random misdetection and can be improved by filtering the position of the ball between frames.

VII. CONCLUSIONS AND FUTURE WORK

Our efforts to develop a low-cost integrated system for tennis ball retrieval have thus far resulted in the iterative development of a tested, proven hardware platform. The software stack has been developed for court localization, navigation, and ball detection. We are still working on the robustness of court localization and further code optimizations, which are two necessary steps for the integration of these components.

The eventual goal for this project is fully automated ball retrieval for tennis matches. We are encouraged by the preliminary results for localization, motion planning and ball detection. While we continue to make the system robust to environmental variations, we also aim to develop the decision-making functionality of the platform to create a truly autonomous system.

VIII. ACKNOWLEDGEMENTS

This project was developed by a team which will hopefully carry the project onto completion of its goals. I would like to acknowledge Karthik Lakshmanan for his work in motion planning, Ankush Gupta for his work in constructing the ball pickup and developing the ball detection algorithm, and Pieter Abbeel for his overall guidance and support in overseeing the project.

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.