20th RoboCup International Symposium, Leipzig, Germany, July 2016.

Virtually Adapted Reality and Algorithm Visualization for Autonomous Robots

Danny Zhu and Manuela Veloso

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract. Autonomous mobile robots are often videotaped during operation, whether for later evaluation by their developers or for demonstration of the robots to others. Watching such videos is engaging and interesting. However, clearly the plain videos do not show detailed information about the algorithms running on the moving robots, leading to a rather limited visual understanding of the underlying autonomy. Researchers have resorted to following the autonomous robots algorithms through a variety of methods, most commonly graphical user interfaces running on offboard screens and separated from the captured videos. Such methods enable considerable debugging, but still have limited effectiveness, as there is an inevitable visual mismatch with the video capture. In this work, we aim to break this disconnect, and we contribute the ability to overlay visualizations onto a video, to extract the robot's algorithms, in particular to follow its route planning and execution. We further provide mechanisms to create and visualize virtual adaptations of the real environment to enable the exploration of the behavior of the algorithms in new situations. We demonstrate the complete implementation with an autonomous quadrotor navigating in a lab environment using the rapidly-exploring random tree algorithm. We briefly motivate and discuss our follow-up visualization work for our complex small-size robot soccer team.

1 Motivation and introduction

Imagine watching a mobile autonomous robot, or a video of one, as it moves about and performs some task in the world, and trying to infer what it intends to accomplish. Seeing such a robot can be interesting, but only a tiny amount of the information contained in and used by the algorithms that control the robot is actually available in this way.

Having some means to expose this hidden state of robots in an intuitive manner is valuable for both the developers of the robots and for other observers. Typical debugging and information displays show some abstract version of the state on a screen, ranging in level of detail from simple text output to two-dimensional displays to full three-dimensional renderings of the robot. Such displays can be informative, but there remains a visual mismatch with reality: any display is still disjoint from the actual view of the robot itself. By merging the depictions of the real robot and the debugging information, we can break this disconnect and obtain a much better view of the progress of the algorithm. We contribute the ability to create a view of a robot in motion that takes a plain video and combines it with logs made during execution by adding extra drawings on top of the video; these drawings depict extra information to give direct insight into the execution of the algorithm.

In this work, we chose to demonstrate these drawing techniques using navigation in a two-dimensional environment with a quadrotor, which is shown in Figure 1. We define sets of obstacles in the space and instruct the quadrotor to fly between specified points while avoiding the obstacles. We take visualizations generated by the control algorithms and draw them with the correct perspective and occlusion onto a video of the quadrotor such that they appeared in the video to be markings on the ground. The quadrotor is controlled by algorithms intimately tied to the robot's location and the space around it, but with a great deal of computation going on to produce the result at each step which cannot be understood solely from the original video. Creating augmented visualizations is especially helpful when working with robots such as quadrotors, which have the capability to move in ways that humans cannot follow.

Besides revealing internal details of the planning done by the robot, we also demonstrate the creation and display of virtual adaptations of the real environment, in the form of obstacles which are present only virtually; without the integrated display that we provide, there would be no way to see such items in their context relative to the execution of the algorithm.

A primary motivation of our work comes from our experience with CMDragons, our robot soccer team for the RoboCup Small Size League (SSL) [12, 13]; we have created similar visualizations for our SSL team. The control programs of teams in the SSL are complex, with hierarchical architectures containing dozens of subcomponents, each making decisions based on the state of the world 60 times per second. While videos of SSL are interesting to watch, the fast pace and small game objects can make it difficult to tell in detail what is happening on the field at any given moment. In Figure 2, we show a typical frame from a video of an SSL game, along with the result of adding our drawings on top of it.

Fig. 1. The quadrotor we used to demonstrate our visualization, alongside its protective hull, which is used for safety when flying indoors. The colored pattern on top is not normally part of the hull; we used it for tracking (see Section 5).

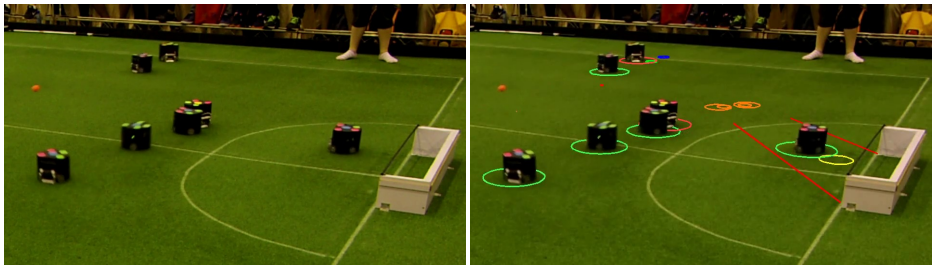

Fig. 2. Left: a plain frame from a video of the SSL. Right: an example visualization created based on that frame. Robots of the two teams are surrounded by circles of different colors, and other drawings generated by our team code are projected onto the field as well.

2 Related work

The visualization we present here is closely related to augmented reality, which is, broadly, the inclusion of virtual objects in the view of a real 3-D environment. Augmented reality has seen a wide variety of uses, both in relation to robotics and otherwise; it allows for enhancement of a user's view of reality by providing additional information that would not otherwise be available. Azuma [2] listed a variety of uses of augmented reality, including those for medicine, manufacturing, robotics, entertainment, and aircraft control. One common use of augmented reality within robotics is to provide enhanced interfaces for teleoperated robots. Kim [10] described an interface for controlling a robotic arm that could overlay 3-D graphics onto a live video view of the arm. The interface provided a view of the predicted trajectory of the arm, allowing for much easier control in the presence of signal delay between the operator and the arm. Our work shares the 3-D registration and drawing that are common to augmented reality systems, although most definitions of augmented reality require an interactive or at least live view, which we do not currently provide. Amstutz and Fagg [1] developed a protocol for communicating robot state in real time, along with a wearable augmented reality system that made use of it. They demonstrated an example application that overlaid information about nearby objects onto the user's view, one that showed the progress of a mobile robot navigating a maze, and one that showed the state of a robotic torso. Though they focused mainly on the protocol aspect, their work has very similar motivations and uses to ours.

Other recent work has also involved using augmented reality overlays to aid in the understanding of the behavior of autonomous robots. Chadalavada et al. [6] demonstrated a robot that projected its planned future path on the ground in front of it. They found that humans in the robot's vicinity were able to plan smoother trajectories with the extra information, and gave the robot much higher ratings in attributes such as predictability and reliability, verifying that describing an agent's internal state is valuable for interacting with humans. Collett [7] developed a visualization system similar to the one described here, but we focus on combining the drawing with the ability to display and animate details of the future plan of the robot, rather than the low-level sensor and state information from the robot.

Animation has long been an invaluable tool for understanding algorithms. In algorithm animation, the changes over time in the state of a program are transformed into a sequence of explanatory graphics. Done well, such a presentation can explain the behavior of an algorithm much more quickly than words. BALSA [4] is an influential early framework for algorithm animation. It introduced the idea of "interesting events," which arise from the observation that not every single operation in an algorithm should necessarily be visualized. In BALSA, an algorithm designer inserts calls to special subroutines inside the algorithm; when execution of the algorithm reaches the calls, an interesting event is logged, along with the parameters to the call. The designer then writes a separate renderer that processes the log of interesting events into an actual animation to display.

We are performing a form of algorithm animation here, but with an unusual focus: most animations are designed with education in mind and delve into the details of an abstract algorithm, rather than being integrated with a physical robot. Additionally, the typical lifecycle of an algorithm being executed on a mobile autonomous robot is different from the standalone algorithms that are usually animated. In most animations, a single run of the algorithm from start to finish results in an extended, and each interesting event corresponds to some interval of time within it. For an autonomous robot, algorithms are instead often run multiple times per second, with each run contributing a tiny amount to the behavior of the robot, and it makes sense for each frame of an animation to depict all the events from one full run. An example of this kind of animation comes from the visual and text logging systems employed by teams in the RoboCup Small Size League, a robot soccer league. The algorithms behind the teams in the league consist of many cooperating components, each with its own attendant state and computations running 60 times per second. As a result, understanding the reasons behind any action that the team takes can be challenging, necessitating the development of powerful debugging tools in order for a team to be effective. For text-based log data, Riley et al. [14] developed the idea of "layered disclosure," which involves structuring output from an autonomous control algorithm into conceptual layers, which correspond to high- and low-level details of the agent's state. Teams have also developed graphical logging in tandem with the text output: the algorithms can output lists of objects to draw, and we can view both this drawing output and text output either in real time or in a replay fashion. The existence of these tools speaks to the need for informative debugging and visualization when dealing with autonomous agents.

3 Pipeline

The overall sequence of events in creating the visualizations we demonstrate is as follows. First, we record a video of the robot or robots while the algorithm is running. During execution, each run of the algorithm saves a sequence of interesting events to a file, along with the current time.

Then we manually annotate certain aspects of the video to enable the rendering to be done: the time at which the video started, which allows each frame of the video to be corresponded with the interesting events from the appropriate run of the algorithm, and the ground and image coordinates of at least four points on the ground in the video,

which allows any point in the ground coordinates (with which the algorithm) works to be translated into the corresponding coordinates in the video. We only need to mark points on the ground plane because we are only drawing points within it.

Finally, we combine the video, the interesting events, and the annotations to produce one output video containing both the video recording and extra information. During this stage, each frame of the video of the real robot is associated with the events from the most recent run of the algorithm. Each event is then transformed into zero or more appropriate drawing primitives, which are overlaid onto the frame as described in Section 4.

4 Rendering

To take the descriptive information generated by the algorithm and draw it convincingly on the video, we need two key pieces of information for each frame: the set of pixels in the frame that should be drawn on, and the transformation from the ground coordinates used by the algorithm into video pixel coordinates.

4.1 Masking

We were flying our quadrotor above an SSL playing field, and we wanted to draw only on the field surface, not on top of the quadrotor or any obstacles on the field. In order to do so, we need to detect which pixels in the video are actually part of the field in each frame. Since the majority of the field is solid green, a simple chroma keying (masking based on the color of each pixel) mostly suffices. We convert the frame to the HSV color space, which separates hue information into a single channel, making it more robust to lighting intensity changes and well-suited for color masking of this sort. We simply took all the pixels with hue values within a certain fixed range to be green field pixels, providing an initial estimate of the mask.

To account for the field markings, which are not green, we applied morphological transforms [15] to the mask of green pixels. The idea is that the markings form long, thin holes in the green mask, and there are no other such holes in our setup; therefore, a dilation followed by an erosion with the same structuring element (also known as a closing) fills in the field markings without covering any other pixels. We also applied a small erosion beforehand to remove false positives from the green chroma keying. The structuring elements for all the operations were chosen to be iterations of the cross-shaped 3×3 structuring element, with the number of iterations chosen by hand.

4.2 Coordinate transformation

Since we are only concerned with a plane in world space, we need a homography to give the desired coordinate transformation, if we assume an idealized pinhole camera [8]. A pinhole camera projects the point (x, y, z) in the camera's coordinate system to the image coordinates $\left(\frac{x}{z}, \frac{y}{z}\right)$, so the coordinates (u, v) are the image of any point with coordinates proportional to $(u, v, 1)$. Suppose that the ground coordinates $(0, 0)$ correspond to the coordinates $\boldsymbol{q}$ in the camera's coordinate system, and that $(1, 0)$ and $(0, 1)$ correspond to

$\boldsymbol{p}_x$ and $\boldsymbol{p}_x$ respectively. Then, for any x and y, the ground coordinates (x, y) correspond to the camera coordinates

$$x\,\boldsymbol{p}_x + y\,\boldsymbol{p}_y + \boldsymbol{q} = \begin{pmatrix} \boldsymbol{p}_x & \boldsymbol{p}_y & \boldsymbol{q} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Thus, multiplying by the matrix $\begin{pmatrix} \boldsymbol{p}_x & \boldsymbol{p}_y & \boldsymbol{q} \end{pmatrix}$ takes ground coordinates to the corresponding image coordinates; the resulting transformation is a homography. There are well-known algorithms to compute the homography that best fits a given set of point or line correspondences. We used the function for this purpose, `findHomography`, from the OpenCV library [3].

We primarily used a stationary camera; for such videos, we manually annotated several points based on one frame of the video and used the resulting homography throughout the video. We used intersections of the preexisting SSL field markings as easy-to-find points with known ground coordinates. We also implemented a line-tracking algorithm similar to the one by [9], which allows some tracking of the field with a moving camera.

4.3 Drawing

Finally, with the above information at hand, drawing the desired shapes is straightforward. To draw a polygon, we transform each vertex individually according to the homography, which gives the vertices of the polygon as it would appear to the camera. Since a homography maps straight lines to straight lines, the polygon with those vertices is in fact the image of the original polygon. Then we fill in the resulting distorted polygon on the video, only changing pixels that are part of the field mask. We used crosses and circles (represented as many-sided polygons) as primitives. Figure 3 shows an example of an frame from the output video, along with the image created by drawing the same primitives directly in a 2-D image coordinate system, and Figure 4 demonstrates the stages involved in drawing each frame.

5 Quadrotor navigation

As our testbed for demonstrating this merged visualization, we used the rapidly-exploring random tree (RRT) algorithm [11] to navigate a quadrotor around sets of virtual and real obstacles.

5.1 The RRT algorithm

The RRT algorithm finds a path from a start state to a goal state within some state space, parts of which are marked as obstacles and are impassable. The most basic form of the algorithm consists of iterating the following steps, where the set of known states is initialized to contain only the given start state:

1. generate a random state r

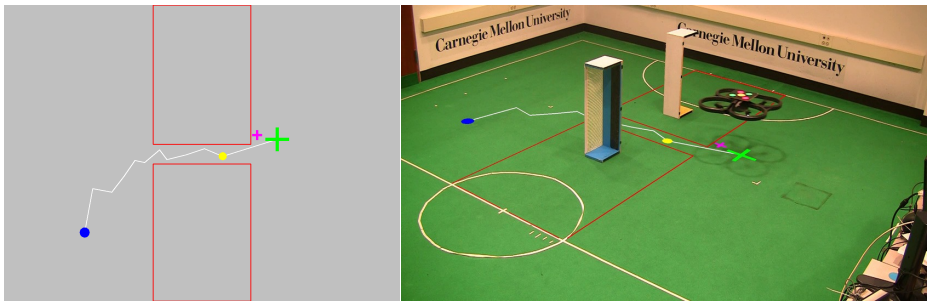

Fig. 3. Left: A 2-D visualization generated from the interesting events of one run of the RRT, using a domain based on physical obstacles. Right: The result of drawing that visualization onto the corresponding frame from the video of the real robot.

Fig. 4. The stages in processing each frame of a video to overlay the virtual drawings. From left to right: (1) the original frame, (2) the raw green mask, (3) the mask, after morphological operations, (4) the result of drawing without taking the mask into account, and (5) the drawing, masked appropriately.

Fig. 5. The visualization generated from the RRT running with a domain consisting of entirely virtual obstacles.

2. find the closest known state c to r
3. extend c toward r, creating e
4. if e is directly reachable from c, add e to the set of known states, recording that c is its parent

The iteration terminates when a state is found sufficiently close to the goal state. The entire path is then constructed by following the chain of parents to the start state.

The simplest state space for an RRT consists of the configuration space of the robot, with dynamics ignored; states are connected simply by drawing lines in the space. Although this is a great simplification, it is straightforward in concept and implementation, and often leads to sufficiently good results.

Since the quadrotor can accelerate in any direction without yawing to face it, we chose to ignore orientation and take states to be the locations within a rectangular region on the ground. As is typical for kinematic RRTs, the random state generation simply chooses a location uniformly at random within the set of configurations and the metric is Euclidean distance between locations.

A common optimization used with RRTs is to simplify the returned path by skipping directly from the start state to the last state in the path which is directly reachable from it [5]. This causes the robot to navigate directly toward the first turn on the path; without it, the randomness of the RRT means that the first step in the path may take the robot in widely varying directions between timesteps. After each run of the RRT, the quadrotor attempts to fly toward the point resulting from this optimization.

5.2 Obstacle domains

For simplicity of implementation, our obstacles consisted only of straight line segments (which can be used to build polygonal obstacles). Some of the domains were based on real objects, as shown in Figure 3, while some were purely virtual, as shown in Figure 5.

The purely virtual domains provide a particularly interesting use case for our kind of visualization. Without this augmented display, the perceived behavior of the robot is hard to explain: it simply follows a convoluted path while in apparently open space. As always, information about the algorithm can still be displayed disjointly from the quadrotor itself, e.g., in a two-dimensional visualization on a screen, but the combined animation provides reduced indirection and a more intuitive presentation.

5.3 Interesting events

Conceptually, we think of each interesting event as containing an event type, the possibilities for which are described below, along with a list of parameters, whose meanings depend on the type.

Most of the interesting events during one full run of the RRT roughly correspond to the individual steps of the iteration. They fall into the following categories:

- the generation of a random state,
- the addition of a new known state,
- the addition of a state to the final path, and

– the computation of the first state of the simplified path.

For the first and last category, the parameters simply contain the coordinates of the state in question; for the second and third, the parameters contain the coordinates of the state and its parent.

We also treat the following items as events in that they represent information that is useful for monitoring the algorithm and can lead directly to drawings; although they are perhaps not "events" in the usual sense:

– the locations of the obstacles,
– the start state (i.e., the current position of the robot),
– the goal state, and
– the acceleration command sent to the quadrotor.

After the fact, during the processing of the video, we can choose which of the types of events to include in the processing of the video, and how to depict each one. In the examples shown here, we draw lines from the first simplified state to its parent (which is the start state), and from all subsequent states in the path to their parents. We also distinguish the first simplified state and goal state with circles and the start state with a cross. The acceleration command is depicted with a smaller cross whose position is offset from the start state by a vector proportional to the acceleration. Finally, the obstacles are drawn as red lines in every frame.

5.4 Sensing and planning

In order to provide high-quality tracking of the quadrotor's position, we attached an SSL-Vision [16] pattern to the top of the quadrotor. SSL-Vision uses overhead cameras to track the locations of specific colored patterns; it is typically used for the RoboCup SSL, which has robots solely on the ground, but is flexible enough to track the pattern without modification despite the tilting of the quadrotor and the increase in perceived size due to its altitude.

The main control loop executes each time sensor data are received from the quadrotor, which occurs at approximately 15 hz. Using the position of the camera and the reported altitude from the sensor data, it computes a true position above the ground. (Since SSL-Vision is intended for robots at fixed heights, the position it reports is actually the intersection of the line from the camera to the quadrotor with a horizontal plane at the robot height.) We estimate the current velocity of the quadrotor by performing a linear fit over the last five observed positions.

At each iteration, the planner runs the RRT from the current position to the current goal to generate a new path through the environment. Only the first position in the optimized path is relevant; the controller attempts to fly the quadrotor toward that position.

Since the obstacles in the state space are virtual, it may happen that the quadrotor moves across an obstacle, between states that are not supposed to actually be connected. When this happens, the planner ceases normal path planning and moves the quadrotor back to the point where the boundary was crossed, to simulate being unable to move through the obstacle. In this case, the last valid point is shown with a red cross, as in Figure 6.

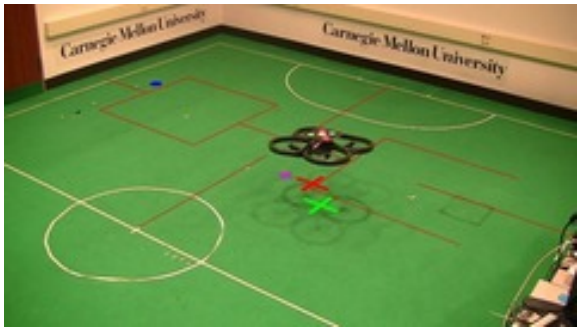

Fig. 6. The visualization generated when the quadrotor has executed an impossible move across an obstacle. The target point is the circle in the top corner, but navigation cannot continue until the quadrotor returns a point reachable from the red cross.

5.5 Control

There remains the problem of actually moving the quadrotor to the target point. The quadrotor accepts pitch and roll commands, which control the horizontal acceleration of the quadrotor, as well as yaw and vertical speed commands, which we use only to to hold the yaw and altitude to fixed values. Since acceleration can only be changed by rotating the whole body of the quadrotor, there is latency in the physical response, compounded by the latency in the vision system and communication with the quadrotor.

Thus we need to supply acceleration commands that allow the quadrotor to move smoothly to a desired point in the face of latency and unmodeled dynamics. Since the low-level control was not the focus of this work, we simply devised the *ad hoc* algorithm shown in Algorithm 1. It takes the displacement to the target from a projected future position of the quadrotor, computes a desired velocity which is in the direction of the displacement, and sets the acceleration to attempt to match that velocity. The perpendicular component of the velocity difference is more heavily weighted, since we considered it more important to get moving in the right direction than at the right speed.

6 Conclusions and future work

In this paper, we demonstrated a means of visualizing the algorithms controlling an autonomous robot that intuitively depicts the relationship between the robot's state and the environment around it. We overlaid drawings generated based on the navigation algorithms for a quadrotor such that they appear to be part of the environment itself.

We intend to extend and generalize the implementation of the interesting event handling and drawing to other domains. A first step in that direction is the SSL drawings discussed previously; we have implemented the equivalent drawings for the SSL, but for the moment, the two systems are disparate and ad hoc, and we would like to unify the representations and interfaces involved.

Algorithm 1 The algorithm used to control the quadrotor. The current location of the quadrotor is denoted by loc, its velocity by vel, and the target point by $target$; we define $\mathrm{proj}_u\, v$ to be the projection of v onto u and $\mathrm{bound}(v, l) = \frac{v}{||v||} \cdot \min(l, ||v||)$.

$\Delta t_{loc} = 0.3\,\mathrm{s}$
$\Delta t_{vel} = 0.5\,\mathrm{s}$
$vel_{max} = 1000\,\mathrm{mm/s}$
$a_{max} = 1500\,\mathrm{mm/s}$
$\Delta t_{acc}^{\parallel} = 0.8\,\mathrm{s}$
$\Delta t_{acc}^{\perp} = 0.5\,\mathrm{s}$
function CONTROL(loc, vel, $target$)
    $loc_{fut} \leftarrow loc + \Delta t_{loc} \cdot vel$
    $\Delta loc \leftarrow target - loc_{fut}$
    $vel_{des} \leftarrow \mathrm{bound}\left(\frac{\Delta loc}{\Delta t_{vel}}, vel_{max}\right)$
    $\Delta vel \leftarrow vel - vel_{des}$
    $\Delta vel_{\parallel} \leftarrow \mathrm{proj}_{vel_{des}} \Delta vel$
    $\Delta vel_{\perp} \leftarrow \Delta vel - \Delta vel_{\parallel}$
    return $\mathrm{bound}\left(\frac{\Delta vel_{\parallel}}{\Delta t_{acc}^{\parallel}} + \frac{\Delta vel_{\perp}}{\Delta t_{acc}^{\perp}}, a_{max}\right)$
end function

We would also like to improve the interactivity of the system as a whole; although the video processing already occurs at faster than real time, equipment limitations mean we need to stop recording before beginning processing. We are interested in allowing the robot to respond to actions by a human viewing the visualization, such as changing the target point or the set of obstacles, creating a closed loop system between human and robot that incorporates both physical and virtual elements.

For now, the capabilities of our system with regards to the input video are also limited: we require a background that is amenable to chroma keying in order to perform the pixel masking. More advanced computer vision techniques could reduce or remove the need for this condition.

Bibliography

[1] Peter Amstutz and Andrew H Fagg. Real time visualization of robot state with mobile virtual reality. In *Proceedings of ICRA'02, the IEEE International Conference on Robotics and Automation*, volume 1, pages 241–247. IEEE, 2002.

[2] Ronald T. Azuma. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.

[3] G. Bradski. OpenCV. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Marc H Brown and Robert Sedgewick. *A system for algorithm animation*, volume 18. ACM, 1984.

[5] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*, volume 3, pages 2383–2388. IEEE, 2002.

[6] Ravi Teja Chadalavada, Henrik Andreasson, Robert Krug, and Achim J Lilienthal. That's on my mind! robot to human intention communication through on-board projection on shared floor space. In *European Conference on Mobile Robots (ECMR)*, 2015.

[7] Toby H. J. Collett. *Augmented Reality Visualisation for Mobile Robot Developers*. PhD thesis, University of Auckland, 2007.

[8] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice-Hall Englewood Cliffs, 2003.

[9] Jean-Bernard Hayet, Justus Piater, and Jacques Verly. Robust incremental rectification of sports video sequences. In *British Machine Vision Conference (BMVC'04)*, pages 687–696, 2004.

[10] Won S Kim. Virtual reality calibration and preview/predictive displays for telerobotics. *Presence: Teleoperators and Virtual Environments*, 5(2):173–190, 1996.

[11] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Iowa State University, 1998.

[12] Juan Pablo Mendoza, Joydeep Biswas, Philip Cooksey, Richard Wang, Steven Klee, and Danny Zhu Manuela Veloso. Selectively reactive coordination for a team of robot soccer champions. In *Proceedings of AAAI'16, the Thirtieth AAAI Conference on Artificial Intelligence*.

[13] Juan Pablo Mendoza, Joydeep Biswas, Danny Zhu, Richard Wang, Philip Cooksey, Steven Klee, and Manuela Veloso. CMDragons 2015: Coordinated offense and defense of the SSL champions. In *RoboCup 2015: Robot World Cup XIX*, pages 106–117. Springer, 2015.

[14] Patrick Riley, Peter Stone, and Manuela Veloso. Layered disclosure: Revealing agents' internals. In *Intelligent Agents VII Agent Theories Architectures and Languages*, pages 61–72. Springer, 2001.

[15] Jean Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.

[16] S. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso. SSL-Vision: The Shared Vision System for the RoboCup Small Size League. In *RoboCup 2009: Robot Soccer World Cup XIII*, pages 425–436. Springer, 2009.