In Proceedings of 7th Workshop on Humanoid Soccer Robots,
IEEE-RAS International Conference on Humanoid Robots, Osaka, Japan, November 2012.

# Comparison of Obstacle Avoidance Behaviors for a Humanoid Robot in Real and Simulated Environments

Stephen G. McGill*, Yida Zhang*, Larry Vadakedathu*,
Aditya Sreekumar*, Seung-Joon Yi*, and Daniel D. Lee*

***Abstract*—RoboCup soccer is still host to frequent collisions between robots that lead to unsavory gameplan that is typically not allowed in human soccer. In the Standard Platform league where ultrasonic sensors are employed, these collisions are forbidden, but there is still leniency in the Humanoid League. Additionally, advanced behaviors, such as dribbling, require robust obstacle avoidance strategies to keep possession of the ball. Challenges in the RoboCup competition seek to demonstrate ways to dribble around static obstacles, but it can be hard to replicate conditions for the sake of iteratively tuning robot behavior. We present a method for evaluating policies in simulation and on the real robot for obstacle avoidance behaviors.**

**Keywords : Obstacle avoidance, robot soccer**

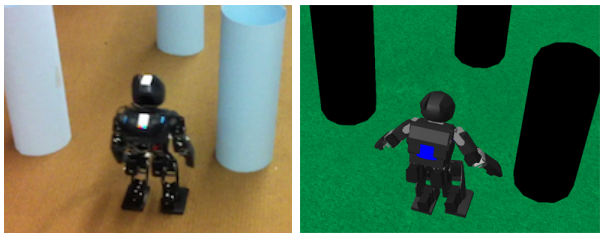


Fig. 1: Navigation through obstacles is tuned in both simulation and on real hardware

## I. Introduction

The RoboCup humanoid soccer competition strives to produce humanoid robots that will someday be able to play soccer against the human soccer team that wins the World Cup. To accomplish this, rules in the league gradually resemble more closely those of FIFA. One of the outstanding issues, however, is that contact between players is not penalized much.

In the Standard Platform League, contact is already banned, but in the Humanoid League, inevitably, this leniency will be removed. It is important to have a framework for preventing contact for current Standard Platform teams and for future Humanoid League teams. Teams have already demonstrated obstacle avoidance and dribbling in the obstacle avoidance challenges extraneous of actual matches. However, this avoidance has not been practically used in real matches. In order to integrate obstacle avoidance and dribbling into real matches, the obstacle avoidance must be proven to be reliable through extensive testing.

One useful testing platform is simulation, where policies are iterated in simulation, and then evaluated on real hardware. We are striving to have real time solution, where particular fail cases on the real hardware can be duplicated immediately in simulation and debugging can happen at the same time on the simulator and the real robot. This requires analysis of the consistency between simulation and real hardware at a high level.

Previous work in analyzing consistencies between simulation and real environment has been examined primarily for motions. If certain sets of parameters yield the same joint angle trajectories on real hardware and simulation, than similar parameters in simulation should also work on real hardware; thus, development can be offloaded from real hardware [1]. However, the relation between simulation and the real robot are never really the same, as shown in comparing a walk engine across simulation and real hardware with the same parameters [2]. This difference can be very pronounced when trying to replicate complex behaviors, since small differences accumulate.

Another approach is to record data from the real robot, and attempt to replicate the fail cases in simulation. The most extensive work at this alignment of simulation and real hardware, for humanoid soccer in particular, comes from the work at Darmstadt [3], where detailed logs are recorded on each robot for offline analysis. We choose to focus on real time debugging in our methods.

Obstacle avoidance systems in RoboCup are not new. From the days of the Aibo, teams have been using vision based obstacle avoidance [4], [5]. With this information, path plans for navigating in and around these obstacles have been considered [6]. However with humanoid robots, there is a larger performance penalty for falling. Additionally, the head can change its gaze in a more pronounced way that can the information available in each frame [7]. Other obstacle detection strategies that do not consider freespace include player recognition [8] and sensing body contact [9] on the B-Human software platform [10]. Outside of the Standard Platform League, however, it is important to detect obstacles without a model of the robot.

In the next sections, we outline our approach, where data is displayed in real time, so that the system can be paused and the world can be analyzed immediately upon detection of a fail case. With logging, it is difficult to the real world conditions that caused a failure.

* GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104
{smcgill3,yida,vlarry}@seas.upenn.edu
{adityasr,yiseung,ddlee}@seas.upenn.edu

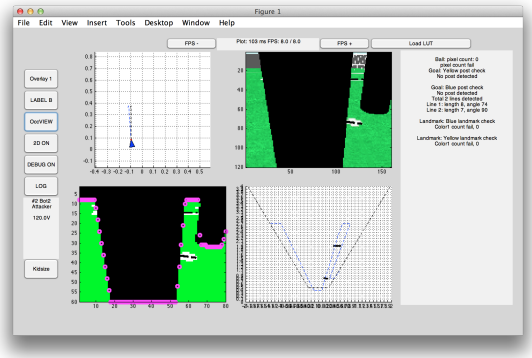


Fig. 2: The real time monitoring GUI provides a good view of the robot's state.

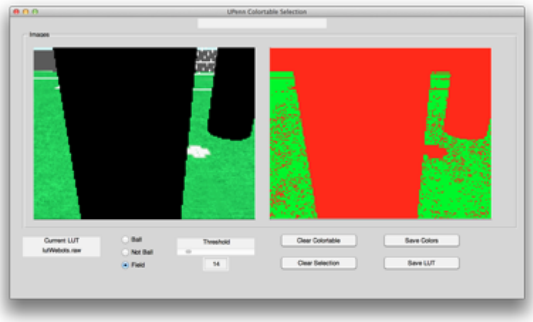


Fig. 3: Human interface for labeling pixels as color classes. Green pixels are ground and red pixels are unclassified.

## II. Debugging Methodology

The overall structure is described in previous work [11]. It consists of motion, vision and behavior subsystems. The vision subsystem processes the video stream and extracts object, such as balls, goalposts and lines. The motion subsystem communicates with robot-specific actuators and sensors, and generates joint trajectories for walking, kicking, and get up routines. The behavior subsystem processes object positions to form a world model and command walking velocities and kicking directions.

All of these subsystems establish a set of shared variables that every other subsystem can access. These variables are also transmitted over a network to a host debugging system, where a human operator can clearly see the state of the robot at any moment. Simulated robots communicate with the host debugging system in a different way to achieve the same goal. At every step of simulation, the simulated robot is able to dump any amount of data on the host computer for immediate debugging.

A real time monitoring program, shown in Figure 2, displays the shared variables to the human user. The user is able to identify any algorithm failures based on either debugging text, or visual cues. The monitoring program displays the same information on the real hardware and simulated hardware using the Webots simulation software [12].

Because of this, we are able to compare the debugging information of the robots when they are placed in nearly the exact same state on a real field and on a virtual field. In this paper, we will outline our approach to debugging obstacle avoidance by iterating between real hardware and simulation.

## III. Vision Consistency

It is crucial that the image processing yields the consistent high level data for the obstacle detection routines in simulation and on the real hardware. We use special real time tools that allow a human user to reduce these differences as they appear.

### A. Color Consistency

Our algorithms rely on lookup tables that translate pixel information (RGB, YUYV, etc.) to discrete color classifications (red, green, orange, etc.). For the simulation to be accurate, similar color classes must be seen at similar perspectives on the real field and the simulated field. The translation from pixels to color classes is done with a trained lookup table.

For obstacle avoidance, we use only two color classes - ground and obstacle. Since only two color codes are used, it is very simple to establish similar colors in simulation and the real hardware. However, the real hardware is susceptible to noise - seeing free space pixels in obstacles, and obstacle pixels in free space.

We mitigate the affects of this noise by tuning the lookup table in real time. We take a live YUV image feed from the robot and display it along with the color classified image. The human user clicks on pixels that it identifies as free space and similar pixels are also selected, with some tunable threshold. Pixels not clicked are identified obstacles. After each click the lookup table is regenerated and sent to the robot.

Shown in Figure 3 is our interface for the human user to identify free space (field). The human user also teaches the ball colors so that dribbling functions can be implemented.

The advantage of this interface is that, in real time, we can update the color classification lookup table on the robot. This is important when lighting changes or when similar colored clothing appears for instance. It is also easy for people to work on the obstacle avoidance code without in depth knowledge of the vision system.

### B. Obstacle Detection Consistency

The colors on a real field will never be classified as perfectly as in the simulator, and so objects will not be detected the same between these platforms. We provide debugging tools to see when and how object detection fails. If failure occurs on the simulator and the real robot, it is a bug in the algorithm, while is failure occurs only on the real robot, it is a tolerance mismatch.

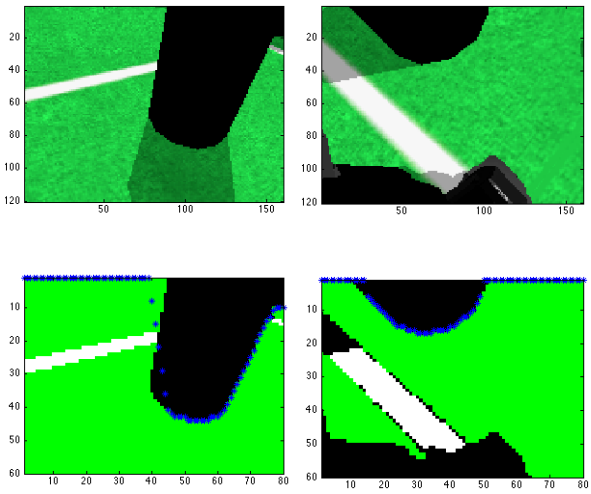
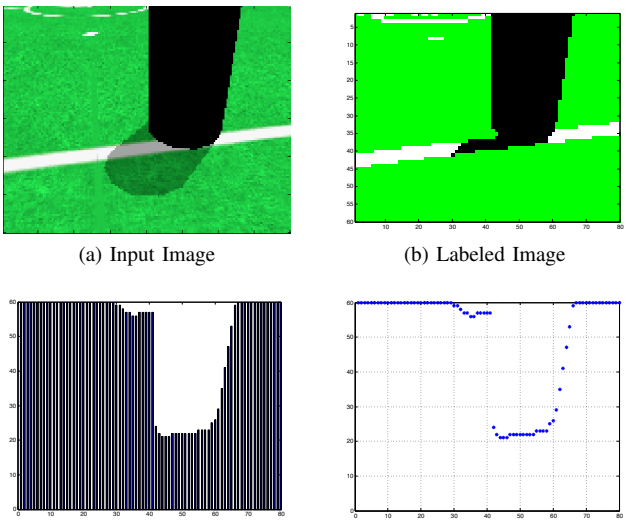


Fig. 4: Failed Contour Detection on the left, and successful contour detection on the right under different head angles and poses.



(a) Input Image (b) Labeled Image

(c) Free Space Pixel Histogram (d) Free Space Contour

Fig. 5: Pixel Histogram based Free space Detection

Without knowing the underlying detection routines, we can display for the user the results of the algorithm. In this way we can position the robot instantly in critical poses and see how the real robot compares to the simulated ones. Again, this debugging allows a user who is unfamiliar with the code to aid in quality assurance. This division of labor is very important for RoboCup teams where personnel can change on a yearly basis.

Shown in Figure 4 is the information displayed to the user. False free space contours and missed boundaries can be reported along with the head angles and the problematic image with a single screenshot.

## IV. Contour Detection Methods

We take a similar approach for detecting contours as outlined on the Aibo [4], where vertical scan lines are used. However, we always scan vertically and then account for the horizon angle at later stages. Two approaches that produce obstacle contours are described below.

### A. Contours from Pixel Accumulation

A first approach for fast computation uses a simple histogram. Using the labeled image $L$, we generate a contour of free space by first vertically accumulating free space pixels. The set of free space colors palette $C$ are defined in binary for orange ($000010_B$), green ($010000_B$), and white ($100000_B$) representing the ball, field, and lines, respectively, in a RoboCup competition.

Intuitively, we generate free space pixel histogram for each column, denoted $F(c)$, which is visualized in Figure 5c. By simply picking the value of every histogram bin, we have the free space contour as Figure 5d.

$$F(c) = \sum_r [L(r,c) \in C] \quad (1)$$

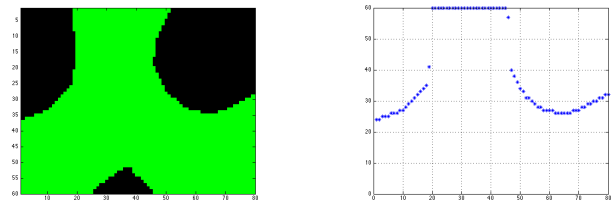
where the number of labeled pixels in the set of free space pixels is summed over all rows $r$ for a column $c$.

### B. Contours from Clustering

A more computationally intense procedure using clusters of obstructing pixels is robust to some of the failure cases seen in the histogram method. In this method, we cluster all the labeled non-freespace cells into group, and analyze some of their important properties. These clusters are performed column by column.

| Property | Description |
|---|---|
| $\text{CENT}_{c,i}$ | The Centroid of the Cluster |
| $\text{SIZE}_{c,i}$ | The Number of Pixels in the Cluster |
| $\text{LOW}_{c,i}$ | The Lower Bound of the Cluster |
| $\text{UP}_{c,i}$ | The Upper Bound of the Cluster |

TABLE I: Properties of Clusters with Non-Free Space Color



(a) Noise-Free Labeled Image (b) Free Space Contour

Fig. 6: Pixel Clustering based Free space Detection

This method allows us to tune performance for robustness to noisy pixels. Clusters with small pixels counts are ignored when they go below a certain threshold. Clusters that appear too low in the image are also ignored.

## V. World Coordinate Projection

Given any contour producing method, we project the output contour into world coordinates for use in avoidance behaviors. The free space contour CONT$(I_x, I_y)$ is detected in the image coordinate frame in terms of pixels. We consider the problem of projecting the pixel based free space contour to robot body coordinate as CONT$(x_b, y_b, z_b)$.

Using a monocular camera mounted on a set of servos, we must make the assumption that all pixels seen in the frame occupy the $z = 0$ plane. We can then project all points to $x, y$ space, and transform those $x, y$ points from the frame of the physical camera to the frame of the robot's foot for a practical frame of reference.

We depict the coordinate transform problem in Figure 7. Our goal is to compute the position $P_2$ in the frame $O_B$ when given a pixel point $P_1$ with image coordinates $(x_I, y_I)$. We define the origin of the camera coordinate frame as $P_0$. Points in the camera (non-image) coordinate frame are specified as rays $P_b^1(x_1, y_1, z_1, w)$, where $w$ can constraint the length. Points in the body coordinate coordinate frame are specified as $P_b^2(x_2, y_2, z_2, 1)$.

$$\begin{aligned} P_b^0 &= T_b^c \times P_0 \\ P_b^1 &= T_b^c \times T_c^I P_1 \end{aligned} \tag{2}$$

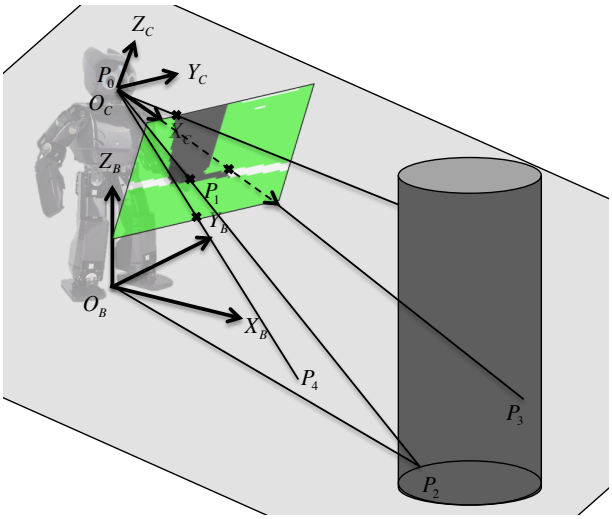



Fig. 7: Projective Transformation Schematic

Consider the ray starting from $P_b^0$ passing through $P_b^1$. It intersects the ground plane $R$ at the point $P_b^2$ with the relation in Equation 3 where $t$ is chosen such that $z_2 = 0$ for $P_b^2$.

$$P_b^2 = P_b^0 - t \cdot (P_b^1 - P_b^0) \tag{3}$$

We project each point along the free space contour in the image coordinate frame, CONT$(I_x, I_y)$ using the described method to generate the free space contour CONT$(x_b, y_b)$ in the robot's body frame. This is computationally tractable for the robot to run at 30fps, since only the boundary pixels are considered.

## VI. Local Contour Tracking

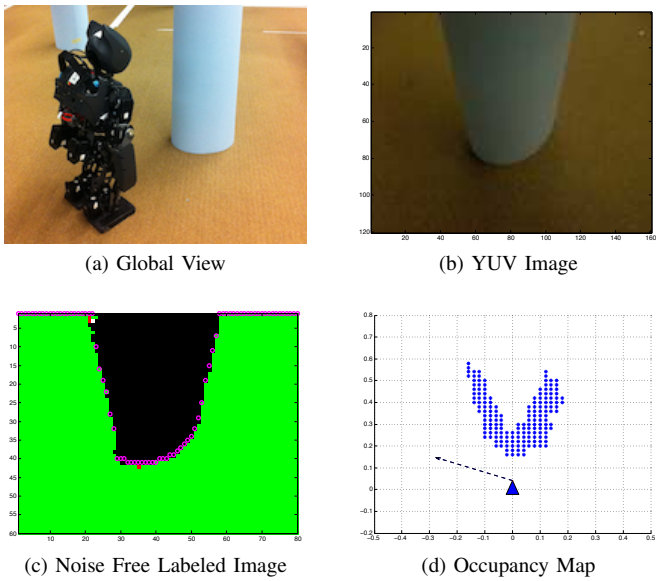
For dynamic environments in which the robot also moves, it is important to keep track of the contour as it changes over time. We initially employed a polar occupancy grid based on previous work [5] to keep track of the local space of the robot, but decided to switch to a clustering approach for a set of discrete obstacles.

### A. Probabilistic Occupancy Grid

We need to account for both odometry and newly perceived contours at each timestep. Our approach is to update the cartesian occupancy map using log odds ratios, where new observations of a grid cell lead to additions, and free space observations of a cell lead to subtractions. Additionally, for every cell that is not observed, there is a decay of the probability that that cell is occupied.

The probalistic approach allows for the map to adaptive to changing environments, since uncertainty in each cell grows over time. This means that our grid is robust to kidnapping of the robot and, potentially, moving obstacles.

An example of this occupancy grid pipeline from image to occupancy grid is shown in Figure 8.



(a) Global View (b) YUV Image

(c) Noise Free Labeled Image (d) Occupancy Map

Fig. 8: The full pipeline of the occupancy grid system including contour detection.

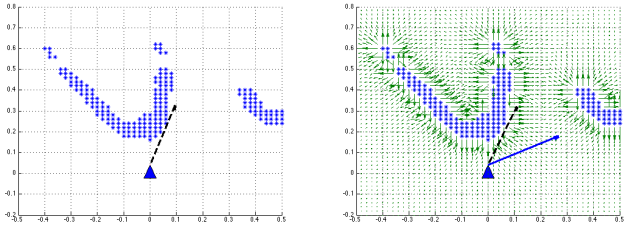
## VII. Obstacle Avoidance Strategies

Given the data from the occupancy map, we can generate a host of obstacle avoidance strategies. It is important to be able to rapidly develop and tune these behaviors for optimal performance in RoboCup. It is very time consuming to do this with real hardware, as opponents must be operated, and fail cases must be replicated often. We thus wish to evaluate policies in simulation as much as possible. Since our vision system can be validated between real hardware

and simulation using the debugging tools described, the next step is to validate the behaviors generated.

### A. Initial Navigation Strategy

Given an occupancy grid of points, we set up a potential field, where the repulsiveness of the field is the summation of all repulsive effects of the occupied grid cells. The center of the goal posts (for our testing strategy) is the lone attractive force, with the occupied points providing repulsive forces.



(a) Occupancy Map (b) Occupancy Repulsion

Fig. 9: Modifying the initial trajectory (Dashed arrow) of the robot based on a potential field to avoid obstacles (solid arrow)

### B. Data Collection

To do this, we compare the paths chosen in identical scenarios on real hardware and the simulated robot. If the paths chosen are similar, then we can expect that behaviors found in simulation will perform the same on real hardware. The scenario includes 10 cylindrical posts spaced .8 meters apart in the configuration shown in Figure 10. There is a pair of green goalposts on one end of the barrier posts, and the robot on the other end.

We log data on an offboard computer from wireless debugging data sent from the robot. We record at 8 Hz the pose of the robot. We show the path points for 3 types of paths – starting from the center, starting from the left corner of the barrier posts, and starting from the right. The tests have been conducted only once to show the working system, however, multiple runs are required for a sound basis of comparison.

In Figure 11, the real robot stopped short of the goal position (3 meters forward, zero meters) in front of the obstacle in front of it. This behavior was observed in the simulator, where there is a large cluster of blue pose points around the (1,-0.5) mark; however, in the simulator, the robot eventually wandered past the barrier. We would like to run more trials to capture this behavior where the robot is "stuck" at an impasse, because it shows how the algorithm is not robust to certain conditions. In other tests, the simulator became stuck but also eventually wandered away, whereas this case from the left was the one instance where the robot in real life stayed in a stuck position for a prolonged period.

In Figure 12, one difference that we saw was that the robot can choose the left or right at the first obstacle. This

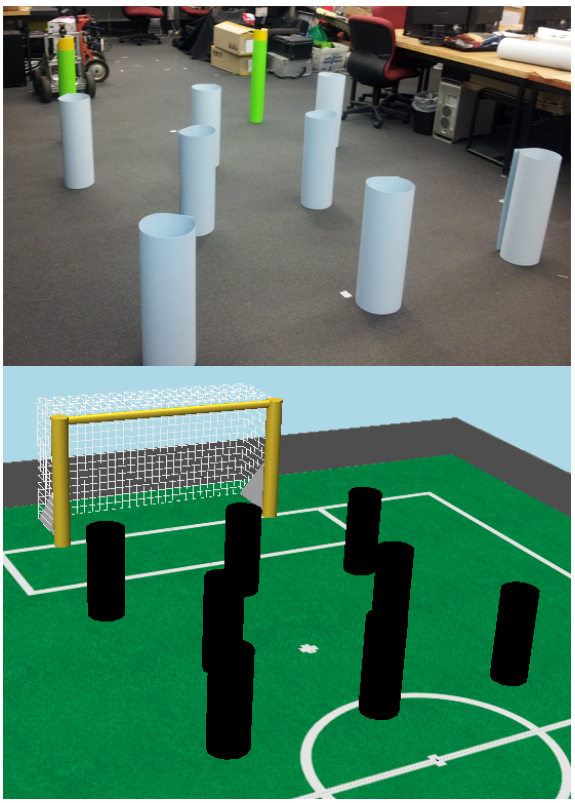


Fig. 10: The configuration of the posts is identical in simulation and for real hardware.

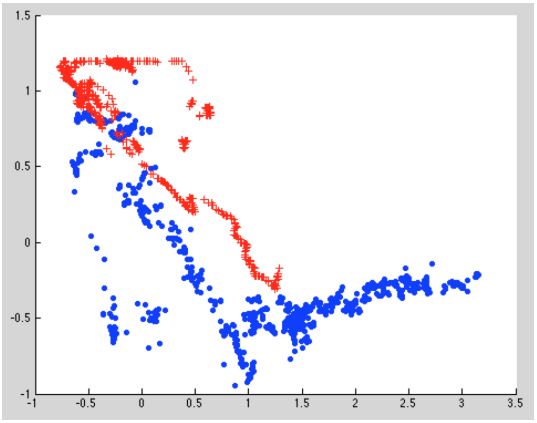



Fig. 11: Path comparison between simulation (blue) and real hardware (red) when starting from the left, where the coordinates are measured in meters and the goal is (3,0)

makes comparison of the path generation difficult, since each path has essentially the same behavior, but there is little overlap. One way to remedy this problem is to reduce the symmetry in the obstacles, as there can be many optimal paths in environments that are symmetric.

In Figure 13, we find that the reported odometry from the real hardware does not match the simulated hardware in scale. This is to be expected, since there can be more

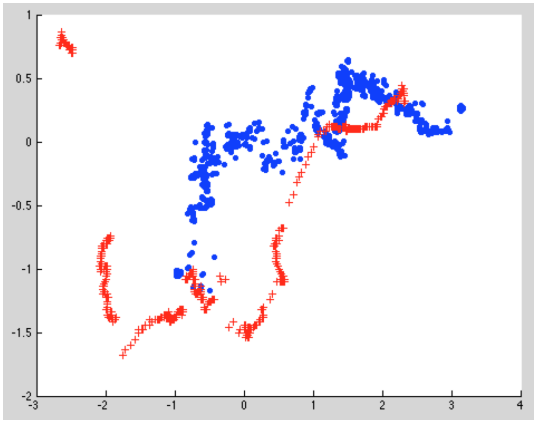



Fig. 12: Path comparison between simulation (blue) and real hardware (red) when starting from the right, where the coordinates are measured in meters and the goal is (3,0)

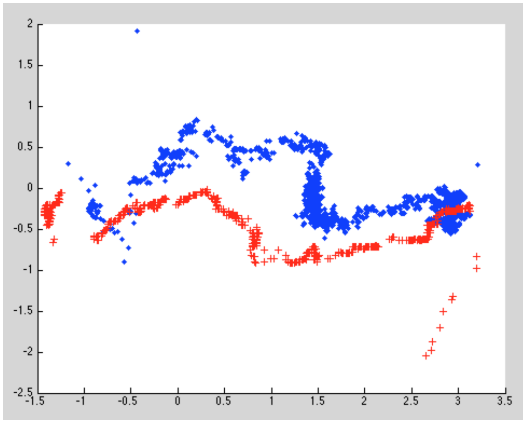



Fig. 13: Path comparison between simulation (blue) and real hardware (red) when starting from the center, where the coordinates are measured in meters and the goal is (3,0)

pronounced localization difficulties in a noisy environment. To mitigate this in future work, we would like to utilize a Vicon system for absolute measurement of the robot's position, rather than rely on the robot to report its own position. The drawback is the time consumption using this approach.

## VIII. Conclusions

With more stringent rules and more realistic play desired, it is imperative that obstacle avoidance become an integral part of the RoboCup landscape. With our framework for evaluating obstacle avoidance strategies in simulation and on the real hardware, more intelligent behaviors can be implemented at a faster rate.

From our initial work, we have identified ways to improve testing conditions for better capturing of data. In addition to collecting data from external sources, we also would like to add more advanced behaviors and test cases where there are moving obstacles. For instance, we plan to integrate ball handling maneuvers into this framework so that dribbling techniques can be employed.

## References

[1] Hirohisa Hirukawa, Fumio Kanehiro, Shuji Kajita, Kiyoshi Fujiwara, Kazuhito Yokoi, Kenji Kaneko, and Kensuke Harada. Experimental evaluation of the dynamic simulation of biped walking of humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1640–1645, 2003.

[2] Shivaram Kalyanakrishnan, Todd Hester, Michael Quinlan, Yinon Bentor, and Peter Stone. Three humanoid soccer platforms: Comparison and synthesis. In *Proceedings of the RoboCup International Symposium 2009*. Springer Verlag, 2009.

[3] Dirk Thomas, Dorian Scholz, Simon Templer, and Oskar von Stryk. Sophisticated ofine analysis of teams of autonomous mobile robots. In *Proc. 5th Workshop on Humanoid Soccer Robots at the 2010 IEEE-RAS Int. Conf. on Humanoid Robots*, Nashville, TN, December 2010.

[4] Jan Hoffmann, Matthias Jngel, and Martin Ltzsch. A vision based system for goal-directed obstacle avoidance. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and Jos Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *LNAI*, pages 418–425. Springer, 2005.

[5] S. Lenser and M. Veloso. Visual sonar: fast obstacle avoidance using monocular vision. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 886 – 891 vol.1, oct. 2003.

[6] Juan Fasola, Paul E. Rybski, and Manuela M. Veloso. Fast goal navigation with obstacle avoidance using a dynamic local visual model.

[7] S. Kohlbrecher, A. Stumpf, and O. von Stryk. Grid-based occupancy mapping and automatic gaze control for soccer playing humanoid robots. In *Proc. 6th Workshop on Humanoid Soccer Robots at the 2011 IEEE-RAS Int. Conf. on Humanoid Robots*, Bled, Oct. 26th - Oct. 28th 2011.

[8] Alexander Fabisch, Tim Laue, and Thomas Rfer. Robot recognition and modeling in the robocup standard platform league. In Changiu Zhou, Enrico Pagello, Sven Behnke, Emanuele Menegatti, Thomas Rfer, and Peter Stone, editors, *Proceedings of the Fourth Workshop on Humanoid Soccer Robots*. o.A., 2010.

[9] Thomas Röfer, Tim Laue, Judith Müller, Colin Graf, Arne Böckmann, and Thomas Münder. B-Human Team Description for RoboCup 2012. In Xiaoping Chen, Peter Stone, Luis Enrique Sucar, and Tijn Van der Zant, editors, *RoboCup 2012: Robot Soccer World Cup XV Preproceedings*. RoboCup Federation, 2012.

[10] Thomas Röfer, Tim Laue, Judith Müller, Alexander Fabisch, Fynn Feldpausch, Katharina Gillmann, Colin Graf, Thijs Jeffry de Haas, Alexander Härtl, Arne Humann, Daniel Honsel, Philipp Kastner, Tobias Kastner, Carsten Könemann, Benjamin Markowsky, Ole Jan Lars Riemann, and Felix Wenk. B-human team report and code release 2011, 2011. Only available online: http://www.b-human.de/downloads/bhuman11_coderelease.pdf.

[11] S. G. McGill, J. Brindza, S.-J. Yi, and D. D. Lee. Unified humanoid robotics software platform. In *The 5th Workshop on Humanoid Soccer Robots*, 2010.

[12] Webots. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

**Acknowledgements**: The authors would like to recognize the National Science Foundation and the Office of Naval Research for partially supporting this work through grants CNS 0958406 and ONR 45006.