To appear in *Proceedings of the RoboCup International Symposium 2011* (RoboCup 2011), Istanbul, Turkey, July 2011.

A low cost ground truth detection system for RoboCup using the Kinect

Piyush Khandelwal and Peter Stone

Department of Computer Science, The University of Texas at Austin
{piyushk,pstone}@cs.utexas.edu

Abstract. Ground truth detection systems can be a crucial step in evaluating and improving algorithms for self-localization on mobile robots. Selecting a ground truth system depends on its cost, as well as on the detail and accuracy of the information it provides. In this paper, we present a low cost, portable and real-time solution constructed using the Microsoft Kinect RGB-D Sensor. We use this system to find the location of robots and the orange ball in the Standard Platform League (SPL) environment in the RoboCup competition. This system is fairly easy to calibrate, and does not require any special identifiers on the robots. We also provide a detailed experimental analysis to measure the accuracy of the data provided by this system. Although presented for the SPL, this system can be adapted for use with any indoor structured environment where ground truth information is required.

Keywords: ground truth, robocup, kinect

1 Introduction

An important prerequisite for most autonomous robot tasks is the ability to self-localize. In order to evaluate self-localization algorithms, the robot's estimate of its location must be compared to the ground truth: its true location. As a result, ground truth data is frequently collected for mobile robots, typically by employing sensors external to the robot. Apart from self-localization, ground truth information can also be used for behavior analysis, by providing the locations of other agents in the environment.

Using a ground truth system requires consideration of a number of factors. These include cost effectiveness, the amount of information that needs to be collected, the accuracy of this information, as well as the ease of use of such a system. These factors typically trade-off against one another. For instance a system constrained by cost may require sacrificing on the amount and accuracy of information. In this paper we present a ground truth system created using the Microsoft Kinect sensor. We apply this system to the domain of robot soccer, namely the Standard Platform League (SPL) from RoboCup. This work was driven with the following constraints in mind.

1. **Low cost -** For obvious reasons, a low cost solution is desirable.

2. **Portability and ease of calibration** - Different space restrictions in different venues dictates that the system should allow for flexible sensor placement. Coupled with a fast calibration time, this allows recording full games between multiple teams at foreign venues.
3. **No special markers on robots** - This is a necessary requirement to record competition games in the SPL, as custom markers are not allowed on the robots.

The Kinect sensor is widely available at a cost of approximately USD 150, and only 2 such sensors are required to span the field completely. This paper describes an open source software infrastructure[1] which will allow RoboCup teams to quickly and easily set up a ground truth system. In this paper, we will also present a detailed experimental analysis of the accuracy of the information that this system provides.

The remainder of the paper is organized as follows. In Section 2, we present a comparison of this system with other related work, and follow with a brief background of the various components of this system in Section 3. We explain our methodology in Section 4 along with an empirical analysis of performance in Section 5. We conclude with an emphasis on directions for future work with a discussion in Section 6.

2 Related Work

Ground truth systems have been used in conjunction with mobile robots both within RoboCup and in other settings. In this section we compare our system primarily with those that have been used within RoboCup. This comparison should also enable the reader to choose the ground truth system best suited to their problem domain.

Our system is similar to the SSL-Vision system [15] in many ways. SSL-Vision is a shared vision system that is used by all teams participating in the RoboCup Small Size League, and is used to estimate the ground truth position of the robots. This system uses 2 RGB cameras to span the entire field, and uses unique identifiers on top of the robots to determine robot locations and orientations. This approach was extended in [13] for the RoboCup Mid Size League. In [11], SSL-Vision was directly used to obtain ground truth positions for the Aldebaran Nao, the robotic platform for the SPL. This was achieved by attaching identifiers on top of the robots' bodies by means of a hardware extension.

An approach requiring the addition of custom hardware to the robot defeats one of the goals of the SPL — to keep a common platform across all teams. Such enhancements are not allowed in competition games, thus limiting such approaches for collecting ground truth information to test environments. Our

[1] Instructions for the download and use of this infrastructure are available at `http://www.cs.utexas.edu/~AustinVilla/?p=research/kinect`

system overcomes the requirement of special markers by utilizing the depth information available from the Kinect. Although we lose orientation information embedded in these markers, we view the gain in the ability to record ground truth information for full games to be advantageous. Additionally, our system is straightforward to implement since it does not require multiple teams to recreate custom extensions to their robots. Note that if orientation information is required, our system can be extended to use special markers as well during testing.

A different approach was used by [9] towards collecting highly accurate ground truth information for a single robot in the SPL. In this approach a motion capture system was used for detecting robot pose by placing a set of 8 identification LEDs on the robot itself. This approach successfully captured a fair amount of information about the robot's pose, including the tilt and roll of the robot body, and the position of the head with respect to the body. The data collected by this approach was made available to the public, which allows teams to run their own localization algorithms on this data-set and test it against the ground truth information. However, to properly evaluate algorithms on the robot, it is necessary to actively collect new ground truth information based on the current behavior and self-localization of the robot. Due to the high cost of a motion capture system, it is impractical for many RoboCup teams to implement such an approach. Additionally, this approach is even more invasive than SSL-Vision in terms of placing additional markers on the robot. In comparison, we believe that our solution has a greater appeal due to its low cost, portability, and ease of use.

3 Background

The Kinect sensor was introduced by Microsoft for the X-Box 360 gaming system. It is is a low cost RGB-D camera, which combines information from a standard CMOS camera with an infrared based depth sensor. It has a horizontal field of view (FOV) of 57 degrees, and a vertical FOV of approximately 43 degrees. The supported depth range from the official specification is 1.2 - 3.5 meters, but in our experiments it has been found to be around 0.7 - 7 meters. The Kinect has already seen some applications to robotics [14], and will be used as a sensor by multiple RoboCup@Home teams for RoboCup 2011 [8] [3]. In our experiments, we have also noted a few drawbacks of this sensor. First, the accuracy of the distance reading is proportional to the distance itself, as the sensitivity of the sensor drops off for larger distances (see Fig. 1b). Secondly, a minimum range of 0.7 meters and an inability to work in direct sunlight makes this sensor unsuitable for some robotics applications.

The software infrastructure of this system builds upon the ROS middle-ware package [10]. We chose to use ROS for two main reasons. First, ROS currently provides 2 drivers for the Kinect, and our system can use either. Second, an implementation of the Point Cloud Library (PCL) [1] is available through ROS. PCL is a relatively new library which provides implementations of a number of

(a) $image_{kinect}$ (b) $pointcloud_{kinect}$

Fig. 1: The image and corresponding pointcloud provided by the Kinect driver in ROS. The colored axes in Fig. 1b give the orientation and location of the coordinate axes for ref_{kinect}

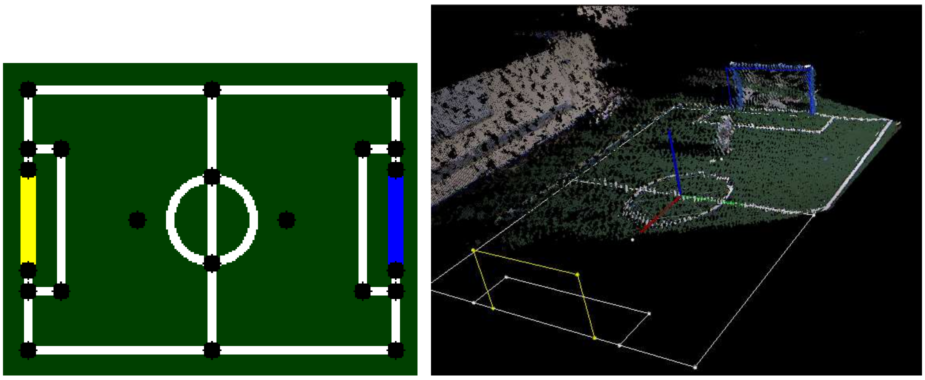
algorithms handling point clouds, some of which have been used in this work. An understanding of ROS may prove beneficial for someone using this system, but it is not entirely necessary.

The Kinect driver from ROS provides information from the Kinect in a number of easily readable formats, and we import this information in 2 formats. The first one is the image that is directly available from the RGB sensor on the Kinect ($image_{kinect}$ in Fig. 1a). The second one is a composite XYZ-RGB point cloud which is created by merging the information from the RGB and depth sensors ($pointcloud_{kinect}$ in Fig. 1b). This point cloud is created by projecting RGB pixels from the camera into 3D space based on the corresponding sensor reading from the depth sensor. The point cloud is provided in a reference frame local to the Kinect (ref_{kinect} in Fig. 1b). ROS also provides sufficient tools for image rectification and geometry to transform a pixel in the RGB image to its corresponding 3D location in the point cloud and vice versa.

4 Methodology

To utilize the information provided by the Kinect, the first necessary step is to obtain the position and orientation of the Kinect sensor with respect to the global coordinate frame of the field (ref_{field}). This global coordinate frame originates at the center of the field, with the $+x$ direction towards the *yellow* goal (see Fig. 3b). Once this information is available, a transformation can be performed on $pointcloud_{kinect}$ to obtain it in the correct reference frame ($pointcloud_{field}$). The calibration procedure to obtain this transformation is explained in Sec. 4.1.[2] To effectively use the color information available in the point cloud, this system also provides a color classification tool, which is briefly explained in Sec. 4.2. Finally, in Sec. 4.3 we explain our methodology for detecting robots and the orange ball.

[2] A video demonstrating calibration, color classification and object detection is available at `http://www.cs.utexas.edu/~AustinVilla/?p=research/kinect`



(a) Landmarks (b) $pointcloud_{field}$

Fig. 2: Obtaining the transformation. Fig. 2a shows the 22 landmarks used for obtaining the transformation, which include all intersections, penalty crosses and goal post bases. Fig. 2b shows the transformed point cloud, along with ref_{field} axes and a wire-frame representation of the field

4.1 Transforming the point cloud

The transformation from ref_{kinect} to ref_{field} involves only rotating and translating the coordinate axes, and is therefore a rigid body transformation. If 2 sets of corresponding points are available in both reference frames, a least squares fitting approach can be used to obtain this transformation even in the presence of noise. A survey of such techniques is provided in [6], and we use the approach given by [2], which uses singular value decomposition (SVD) to calculate this transformation. An implementation of this approach is provided in PCL.

To obtain these corresponding set of points, we make use of some *landmarks* on the field. There are a total of 22 such landmarks, all on the ground plane. The position of these landmarks is already known in ref_{field}, and we provide an interface to the user to provide the corresponding points in ref_{kinect}. For a given landmark, the user clicks a pixel in the image, and the system projects this pixel outwards in 3D space to obtain a ray. We gather points in the point cloud which are close to this ray (within $5cm$), and average over these points to obtain the position of the landmark in ref_{kinect}. If a landmark is not visible by the sensor, then we do not obtain a correspondence.

Directly using this method to obtain landmarks has a drawback. Not all pixels in $image_{kinect}$ have a corresponding point in $pointcloud_{kinect}$. This discrepancy occurs because of slight differences in the field of view of the depth sensor and the rgb camera, as well as the failure of the depth sensor to obtain a reading for some pixels. For this reason it may not be possible to collect information about a landmark even when it is present in the image. To counter this problem, we first ask the user to enter any 5 arbitrary points on the ground plane of the field which are visibly present in the point cloud visualization. With these points, we perform least squares plane fitting [5] to obtain an accurate estimate of the ground plane of the field in ref_{kinect}. We then obtain the pixels corresponding to each landmark in the image and project them as a ray. The intersection of the ray with the ground plane estimate gives the position of the the landmark in ref_{kinect}, and we do not require the landmarks to be at their appropriate positions in the point cloud.

We weigh each correspondence by the inverse of the squared Euclidean distance of the point from the camera sensor (squared L2 norm in ref_{kinect}). The advantage of this weighing is that landmarks which are further away are smaller in the image, and incur a greater error in our estimate of their position in ref_{kinect}. We then apply the procedure described in [2] to obtain the transformation $[\mathbf{R}, \mathbf{T}]$, where $\mathbf{R}$ and $\mathbf{T}$ are the rotation and translation components respectively. We can then construct the transformed point cloud $pointcloud_{field}$ as:

$$point_{field} = \mathbf{R} point_{kinect} + \mathbf{T}$$

$\forall$ $point_{kinect}$ in $pointcloud_{kinect}$. The transformed point cloud is shown in Fig. 2b.

This calibration procedure of obtaining the transformation takes approximately 5 minutes, and needs to be done only once the Kinect has been placed in the desired location. The ability to calculate this transformation into a global coordinate frame also allows for flexible sensor placement, as long as the requisite portion of the field is visible.

4.2 Color Classification

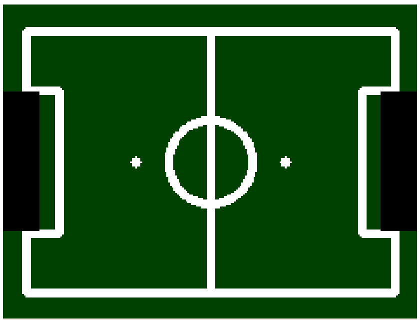
To utilize the color information also available in $pointcloud_{field}$, It is necessary to classify the raw RGB values into known colors of interest. To this end, we use an approach similar to that of many RoboCup teams for performing color based segmentation [12]. Raw RGB values are translated into a color of interest using a 256 * 256 * 256 color look-up table.

To obtain this look-up table, the user picks a pixel in the image corresponding to a color of interest c. Let us say that the RGB value for this pixel is $value_{rgb}$. Based on a sensitivity parameter adjusted by the user, a neighborhood of raw RGB values around $value_{rgb}$ in the color look-up table are then classified as c. After multiple iterations of entering representative pixels for each color of interest, the color table can be used to easily classify either pixels in $image_{kinect}$ or 3D points in $pointcloud_{field}$.

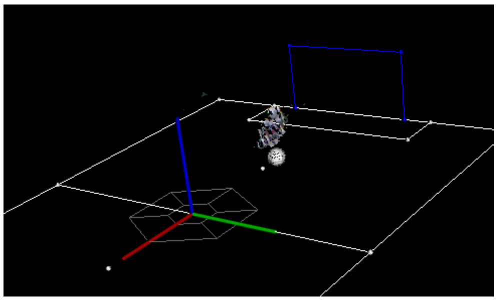
It should be noted that this method typically needs to be performed only once for a given environment and lighting condition, and does not need to be redone with a change in position of the Kinect. Additionally, since the CMOS camera in the Kinect is fairly consistent in terms of colors, this calibration requires a small number of iterations to complete, and takes only a few minutes.

4.3 Object Detection

The point cloud returned by the Kinect driver was found to be robust to large amounts of noise, i.e. no point in $pointcloud_{field}$ was too far away from its true location. This property allows simple heuristics to be used for the object



(a) Robot search region



(b) Detection + $pointcloud_{robot}$

Fig. 3: The robot search region in split up between the 2 Kinect sensors, and each system is run independently. Fig. 3b shows the detection given by our system.

detection task, while at the same time allowing us to obtain the locations of the robots and the orange ball accurately.

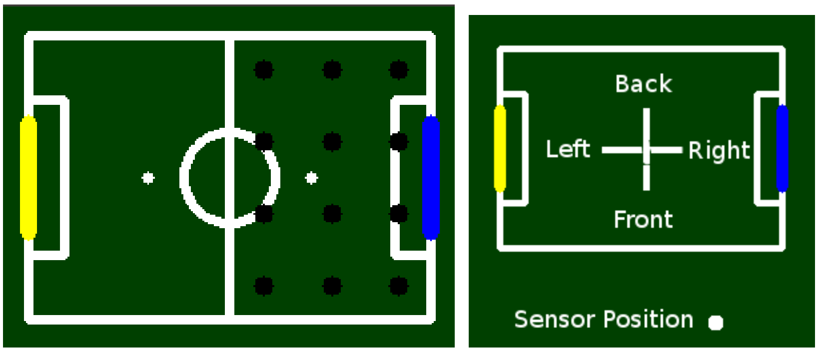
Robot Detection: We define a region in ref_{field} in which we expect to find robots. Since $pointcloud_{field}$ is now available in this coordinate frame, we can filter out all points which could possibly belong to a robot. We construct a new $pointcloud_{robot}$ from these filtered points based on the following constraints:

- We select those points which lie in the region marked out by Fig. 3a. We avoid regions near the goal, since noisy points from the netting or goal post may be detected as a robot.
- We select only those points which are above $30cm$ in height.

Since multiple robots may be present in the region of interest, we apply Euclidean clustering on $pointcloud_{robot}$ in an attempt to obtain 1 cluster per robot. We provide a couple of parameters to the clustering algorithm. The first is a *tolerance* of $10cm$, which defines the maximum distance a point in the cluster may have with its neighbors. Secondly, we retain only those clusters with a minimum size of 200 points (*min_size*). We treat all clusters as possible robot detections, and the x and y coordinates of the cluster centers as the robot's position.

We perform some straightforward error checking for a couple of different cases. First, since it may be required for human referees to step onto the field during a game, we discard any cluster having points above $70cm$ (The nao has a maximum height of $58cm$). Second, if 2 robots on the field are too close, the Euclidean clustering algorithm may return a single cluster for both of them. To avoid a false reading, we also apply some loose size constraints designed to throw out such a cluster. All remaining clusters are reported as robots.

Once a robot has been detected, we attempt to identify its team based on the colored identification marker on its waist. Based on the output of the color classifier, we count the number of *blue* or *pink* pixels close to the cluster center. With this information we assign a team to each detected robot.



(a) Robot positions (b) Robot orientations

Fig. 4: Fig. 4a shows the locations where the robot was placed. Fig. 4b shows the orientation for robot placement relative to the field, along with the position of the sensor.

Ball Detection: Ball detection follows a fairly similar approach to robot detection. We construct a new $pointcloud_{ball}$ from $pointcloud_{field}$ based on the following constraints.

- We select points close to the field, and unlike robot detection do consider the regions around the goals.
- We impose a height constraint for selected points. They should lie fairly close to the ground plane, between a height of $15cm$ and $-15cm$.
- We ensure that only points that have been marked as *orange* are selected.

We perform Euclidean cluster extraction on $pointcloud_{ball}$ as well, using the parameters for *tolerance* of $10cm$ and *min_size* of 10. Finally a size restriction is placed on each cluster, based on the true size of the ball. At this point, if more than 1 cluster still remains, we discard all the readings to prevent any false detection of the ball.

Spanning the entire field: Up to this point, we have only discussed using information from a single Kinect, which can capture information for at most half the field if placed at a reasonable range. To cover the entire field, we split up the field into separate regions, place a Kinect sensor so that it captures one region, and run a separate instance of our system for each Kinect. The search space depicted in Fig. 3a is then split up according to the placement of these regions and provided to each instance of our system. In practice, we have been able to cover the SPL field with 2 Kinect sensors.

5 Experimental Results

In this section, we present experimental results that give error estimates on the information provided by this system. To calculate the error, we place the robot on some known locations on the field (Fig. 4a), and obtain the position returned by our system. The error is then measured by comparing this value against the known value of the location. To avoid being biased by a particular orientation of the robot, we obtain estimates by placing the robot in 4 different orientations at each position. In these orientations, the robot faces the $+y$, $-x$, $-y$ and $+x$ directions as depicted in Fig. 4b. We name these orientations based on their relative positions with respect to the sensor as $front$, $right$, $back$ and $left$, respectively.

We placed the Kinect at a location on the side of the field so that it could sense half the field, and calculated the transformation (see Fig. 4b). Using the transformation, the location of the sensor was recorded at ($-155cm$, $397cm$, $210cm$) in the global field coordinates. We then placed the robot in each of the combinations of positions and orientations one by one, and recorded the output of our system. This recorded reading was averaged across 20 frames captured by the system.

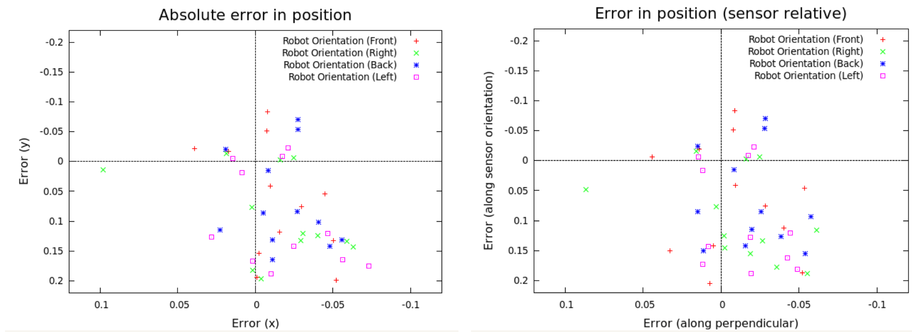
Table 1: Average error in the robot's position.

Type	**Average error** (cm)
Robot (front)	10.19 ($\pm$5.86)
Robot (right)	10.90 ($\pm$5.87)
Robot (back)	9.72 ($\pm$4.55)
Robot (left)	10.87 ($\pm$6.97)
Robot (overall)	**10.41** ($\pm$5.85)

The average error for different orientations of the robot is presented in Table 1. During the collection of this data, there were no false positives (defined as any reading more than $50cm$ away from truth point). The robot was detected in 95.64% of the frames recorded by our system. Detection did not occur in all cases because the Kinect driver occasionally returned an incomplete point cloud, in which the robot was not present.

While performing these tests we came across 2 factors that may add to the reported error of our system. First, while marking the locations of the points shown in Fig. 4a, we realized that due to deformities in the field it was impossible to mark all locations *exactly*. Although these deformities crept into our field through years of use, such errors were also recorded by many teams on the newly constructed venues at RoboCup 2010. As a result, in an attempt to provide realistic error estimates we do not correct for this error. The second error was introduced because of manual placement of robots on these markers, and is unavoidable. It is difficult to measure the amount of error added by these 2 factors, but we do not believe it to to be greater than 2–3cm.

We also measured the consistency of the calibration procedure for obtaining the transformation. For the given Kinect setup, we entered the set of landmarks 3 times and calculated the transformation separately each time. We then selected the 4 outermost landmarks available in the image, and transformed them in the



(a) Robot search region

(b) $pointcloud_{robot}$

Fig. 5: The robot search region is split up between the 2 Kinect sensors, and each system is run independently. Fig. 3b shows the detections given by our system.

ref_{field} coordinate frame using the 3 transformations available. We measured the standard deviation of these points across the 3 calibrations, and found it to be $2.02cm$. This is much less than the noise in our system, suggesting that the calibration procedure produces consistent results.

An interesting observation made from Table 1 was that the errors for the $front$ and $back$ orientations were marginally less than the other orientations. We believe this difference to be a result of a bias introduced by the position of the sensor. Since the points returned by the point cloud are from the surface it hits, these points automatically get moved towards the sensor from the center of the robot. In the $left$ and $right$ orientations the Kinect sensor faces the side of the robot, which is closer to the sensor in comparison of the front or back of the robot. To further investigate the movement of points towards the sensor, we plot the error of each recorded location with respect to its true location in Fig. 5a. The points are biased in the $+y$ direction, which is roughly the same direction in which the sensor lies. To estimate the amount of this shift, we again plot the error, but rotate the axis for each point in the direction of the sensor (Fig. 5b). The mean error along the direction of the camera is $8.15cm$. Knowledge of this value can help compensate for this systematic shift in points, further reducing the error of our system.

6 Discussion

In this paper, we have presented a new ground truth detection system for application in the Standard Platform League of RoboCup. This ground truth system has many desirable qualities such as a low cost, portability, and ease of use, with a fast calibration time. We have also presented an experimental analysis to indicate the system's accuracy. We aim to use this system for evaluating and improving self-localization algorithms on the robots, and expect to find it useful for other applications within the SPL environment as well.

A RoboCup team can easily apply this system to their own research by acquiring the Microsoft Kinect sensor, and using it along with the open source infrastructure that will be made available with this paper. We also hope that as this system finds greater application in the future, members from multiple teams and research labs may contribute towards its further development.

There are still a number of ways in which this system can be improved. Currently this system is unable to provide the orientation of the robot, which is necessary for a full description of the pose as required for self-localization algorithms. Although the solution of appending markers on the robots to determine orientation is possible, an alternative un-invasive method would be preferable. We believe that it may be possible to perform some further analysis of the data available from the Kinect to make some rough estimates of the robot's orientation. For instance, supervised learning techniques may enable shape modeling for estimating orientation based on features obtained from the image and the pointcloud. We hope to examine such possibilities in the future.

Another area that needs further examination is automating the transformation calculation. Although this procedure is fast and requires little input from the user, while conducting our experiments we have occasionally managed to dislodge the sensor from its position requiring recalibration. This is problematic should it happen while recording a competition game, as a few crucial minutes may pass while the sensor is being recalibrated. By taking the user out of the loop, we may be able to avoid such situations. We believe that automated registration [7] may be possible, by registering the point cloud from the Kinect against an artificially generated point cloud of the field. The only requirement is an approach that can scale to the size of the field and the amount of data provided by the Kinect.

Using automated registration has another advantage. Currently we run separate instances of our system for different regions of the field. Typically there is some overlap between the information produced by these sensors, which we currently discard. We do so because unless the sensors are perfectly aligned with respect to each other, slight differences could allow a single robot to be detected twice. By using a technique based on Iterative Closest Point [4], we may be able to register the point clouds against each other with high accuracy.

We believe that this system can also be extended beyond the SPL. Extending it to other RoboCup soccer leagues such as the humanoid soccer league is a fairly straightforward process. In addition, this system should be suitable for any indoor environment which is structured enough to provide a means of obtaining the transformation into a global reference frame. Given constraints about the robots to be detected, the system's parameters can be adapted to provide ground truth regarding their locations over time.

7 Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation

(IIS-0917122), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030).

References

1. Point cloud library. http://www.ros.org/wiki/pcl
2. Arun, K., Huang, T., Blostein, S.: Least-squares fitting of two 3-D point sets. IEEE transactions on pattern analysis and machine intelligence 9(5), 698–700 (1987)
3. Azevedo, J., Cruz, C., Cunha, J., Cunha, M., Lau, N., Martins, C., Neves, A., Pedrosa, E., Pereira, A., Teixeira, A., et al.: Cambada@ home 2011 team description paper. http://www.ieeta.pt/atri/cambada/athome/docs/CAMBADA@Home_TDP2011.pdf
4. Chetverikov, D., Svirko, D., Stepanov, D., Krsek, P.: The trimmed iterative closest point algorithm. Pattern Recognition 3, 30545 (2002)
5. Eberly, D.: Least squares fitting of data. Chapel Hill, NC: Magic Software (2000)
6. Eggert, D., Lorusso, A., Fisher, R.: Estimating 3-D rigid body transformations: a comparison of four major algorithms. Machine Vision and Applications 9(5), 272–290 (1997)
7. Gelfand, N., Mitra, N., Guibas, L., Pottmann, H.: Robust global registration. In: Proceedings of the third Eurographics symposium on Geometry processing. pp. 197–206. Eurographics Association (2005)
8. Jansen, S., Lier, C., Neculoiu, P., Nolte, A., Oost, C., Richthammer, V., Schimbinschi, F., Schutten, M., Shantia, A., Snijders, R., et al.: Borg-the robocup@ home team of the university of groningen team description paper. www.ai.rug.nl/crl/uploads/Site/BORG_TDP_2011.pdf
9. Niemüller, T., Ferrein, A., Eckel, G., Pirro, D., Podbregar, P., Kellner, T., Rath, C., Steinbauer, G.: Providing Ground-truth Data for the Nao Robot Platform. RoboCup 2010: Robot Soccer World Cup XIV pp. 133–144 (2011)
10. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
11. Röfer, T., Laue, T., Müller, J., Burchardt, A., Damrose, E., Fabisch, A., Feldpausch, F., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Honsel, D., Kastner, P., Kastner, T., Markowsky, B., Mester, M., Peter, J., Riemann, O.J.L., Ring, M., Sauerland, W., Schreck, A., Sieverdingbeck, I., Wenk, F., Worch, J.H.: B-human team report and code release 2010 (2010), only available online: http://www.b-human.de/file_download/33/bhuman10_coderelease.pdf
12. Sridharan, M., Stone, P.: Real-time vision on a mobile robot platform. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (August 2005)
13. Stulp, F., Gedikli, S., Beetz, M.: Evaluating multi-agent robotic systems using ground truth. In: In Proceedings of the Workshop on Methods and Technology for Empirical Evaluation of Multi-agent Systems and Multi-robot Teams (MTEE) (2004)
14. Veltrop, T.: Humanoid robot navigation teleoperation using nao and kinect. http://www.robots-dreams.com/2011/01/humanoid-robot-navigation-teleoperation-using-nao-and-kinect-video.html
15. Zickler, S., Laue, T., Birbach, O., Wongphati, M., Veloso, M.: SSL-vision: The shared vision system for the RoboCup Small Size League. RoboCup 2009: Robot Soccer World Cup XIII pp. 425–436 (2010)