

Navigation of Robots in Human Environments

1. Introduction:

Considerable work has already been done in the realm of robot movement in controlled environments, such as factories and workplaces. However, the popularity in sales of domestic robots such as the Roomba®^[4] and Robomow®^[8] suggests a movement towards the realm of household robotics, and therefore a need for the development of accurate control mechanisms.

In any setting, it is important for robots to be able to navigate efficiently from one location to another. Domestic environments, however, provide a new set of challenges. Clutter on the ground can make terrain impassable, moving humans and pets can block pathways, and stairs or uneven surfaces can be damaging.

Several control mechanisms have been explored for such robot control. The iRobot Roomba, for example, follows a bug-like approach—moving forward until an obstacle is hit, and then randomly changing direction^[5]. It utilizes a series of IR depth sensors at its base to detect stairs^[5]. However, a random approach is adequate for only the simplest applications: it does not even attempt to compute the most efficient path to the destination.

The Neato Robotics SV-Series robots utilize an onboard IR depth-sensing array and a SLAM (Simultaneous Localization and Mapping) algorithm to map an area of interest before beginning to clean^[2]. Onboard cameras, however, are limited to the observation of a localized region, and cannot account for obstacles outside of their limited scope. Furthermore, an onboard camera is unable to sense itself, and therefore unable to provide useful diagnostic information other than realizing that it cannot move. Likewise, the robot may have blind spots in its vision when its sen-

Navigation of Robots in Human Environments

sors are too bulky to mount in all possible directions. Interpreting data from multiple sensors can also be too intensive to be carried out on the limited processing power available on robot hardware, while attempting to keep the robot as compact as possible.

This paper proposes an alternative navigation mechanism that addresses the deficiencies in both approaches. An external module is used to map the robot's surroundings, localize the robot within its environment, compute the most efficient path to the destination, and then wirelessly direct the robot along this path. The robot is thereby relieved of doing path computations of its own.

2. Materials & Methods:

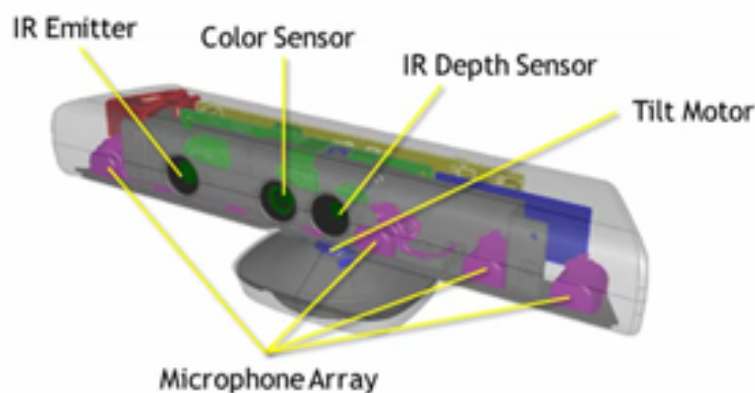


Figure 1 – Diagram of the Microsoft Kinect
Microsoft Corporation. (Introduction to Kinect for Windows Audio) [Diagram], Retrieved 12 September 2013, from: URL (<http://www.microsoft.com/en-us/kinectforwindows/develop/tutorials.aspx>)

This research project utilizes a Microsoft Kinect as an external overhead sensor for mapping and localization.

The Kinect depth sensor works by projecting an irregular dot pattern from the IR emitter. The dot pattern is read through its infrared camera,

and the depth image is calculated based on deviation of the dots^[citation]. The OpenNI libraries that were used to interface with the Kinect do not allow control of the tilt motor or built-in gyroscope. Therefore, the Kinect's angle must be calculated mathematically. The microphone array was not used in this project.

Navigation of Robots in Human Environments

The Orbotix Sphero served as a model household robot—small, agile, and with a distinctive shape. The Sphero contains 3 LED lights (red/green/blue), a 3-axis gyroscope, 3-axis accelerometer, and two motors for locomotion^[citation]. For the purposes of this project, none of these sensors were used. Rather, as described in section 5.1, the Sphero uses its own sensors to maintain a constant *heading* direction regardless of its orientation.



Figure 2 - Image of the Orbotix Sphero
Orbotix. (Sphero Store | Sphero) [Photograph],
Retrieved 01 September 2013, from: URL
(<http://store.gosphero.com/>)

The Kinect can be comfortably used at heights of up to around 11 feet when facing directly downward, and less when at lower angles. The range of the Kinect depth sensor is rated by Microsoft as 4000mm^[6].

For this research project, the Kinect was mounted between 1 and 1.7 meters above the ground, pointing downward at angles ranging between 27 and 45 degrees (Figure 3). The Sphero was placed on a flat, carpeted surface. Obstacles were scattered on the floor.

SimpleOpenNI was used as a Java wrapper to interface with the *OpenNI* Kinect driver. The *Bluecove* library was used to interface with Bluetooth in Java. The *Processing* API was used to handle rendering of the GUI. The Sphero API, with helper libraries (which facilitated communication with the Sphero using Bluetooth) written by Benji Encalada and Nicklas Gavelin, was developed in Java.^[citation?]

2.1 Structure of the Algorithm

The procedure consists of three computational routines, each



Figure 3 - Setup of the Kinect

Navigation of Robots in Human Environments

running simultaneously in its own thread (as described in Figure 4). All three of these threads are created at startup and terminate synchronously on program exit.

Thread 1 is used for obstacle detection and path computation, and relies on depth input from the Sensor. Thread 2 is used for robot control, and relies on the path computed by Thread 1. Finally, Thread 3 is used for robot localization, and relies on the depth and IR input from the Sensor.

The program interfaces with the Kinect through several layers of abstraction. First, the *Processing* API provides function wrappers for SimpleOpenNI, which is a JNI front-end interface for OpenNI—the base C Library. OpenNI relies on the NITE drivers to interface with the Kinect.

The following diagram (Figure 4) illustrates the computational pipeline.

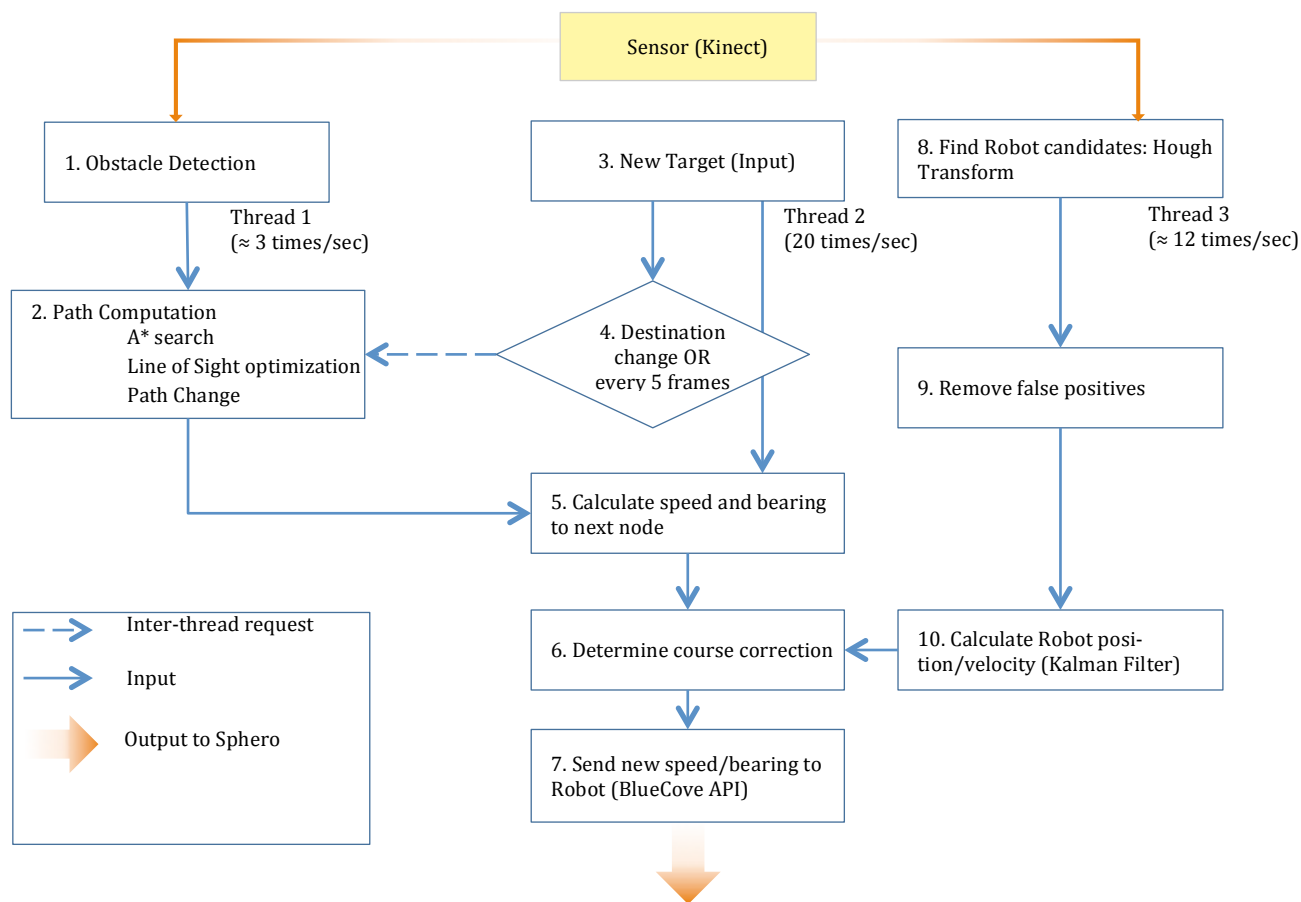


Figure 4: Diagram of Program Flow

Navigation of Robots in Human Environments

1. Obstacles are detected by the Kinect as areas which are greater than a certain height above the ground.
2. Whenever a request is sent by Thread 2 (see step 4), the optimal path from the Sphero's current position to the goal position is calculated using A*.
3. The operator enters a new target in the user interface.
4. When a new target is requested request Thread 1 to compute a path (see Step 2). Approximately every second, request Thread 1 to test the validity of the previously computed path.
5. Calculate the angle and speed the Sphero must follow in order to reach the next node in its path.
6. Use the Sphero's position as detected by Thread 3 to handle the case where the Sphero is straying from its expected path.
7. Send commands to the Sphero via Bluetooth.
8. Use the depth (and possibly IR) image to find Sphero *candidates* (circles) in the scene.
9. Remove *circles* that cannot be the Sphero—such as those which are too high off the ground.
10. Using previous frames, determine the Sphero's position and velocity through a Kalman filter.

3. Robot Detection

3.1 Real-world Coordinate Mapping

The Kinect depth sensor provides a 640x480 array of depth values in millimeters, ranging from 0 to 4000. Depth measurements provided by the Kinect are already corrected for perspective distortion, and therefore represent orthographic distances from the Kinect's view plane, rather than distances from a single point.

The Kinect boasts a field of view of 57° horizontally and 43° vertically^[citation]. Thus, pixel position in the input depth image can be converted to a degree angle measure in the following way:

$$\theta_x = (x_{pixel} - 320) * \frac{28.5}{320}$$

$$\theta_y = -(y_{pixel} - 240) * \frac{21.5}{240}$$

The resulting world coordinates relative to the Kinect are calculated by projecting out in the direction of θ_x and θ_y :

$$\text{Kinect Relative Position } (x, y, z) = (d \sin\left(\frac{\pi\theta_x}{180}\right), d \sin\left(\frac{\pi\theta_y}{180}\right), d).$$

3.2 Floor Plane

RANSAC (RANdom SAmple Consensus) is then used to determine a floor plane^[citation]. RANSAC is an iterative process used to find an accurate model for a set of data—e.g., fitting a line through a set of points. RANSAC excels at determining trend lines through data sets with many outliers, an issue that presents itself when calculating planes of best fit. RANSAC is performed by sampling three random points in the depth image to construct a plane. A confidence value is then created for the plane, which represents how many points lie on it. After 100 iterations, with each iteration checking confidence with 200 points, the plane with the highest confidence value is determined to be the floor plane. This procedure assumes that the floor plane is more prominent than any other equally sized plane in the image, which is generally true.

The Kinect's world angle, in radians, is then calculated. L represents the Kinect's local direction vector, defined as $(0,0,-1)$. N is defined as the normal vector to the floor plane.

$$\theta_{Kinect} = \cos^{-1}(L \cdot N) - \frac{\pi}{2}$$

World position is then defined as:

$$World\ Position\ (x, y, z) = (d \sin\left(\frac{\pi\theta_x}{180}\right), d \sin\left(\frac{\pi\theta_y}{180}\right) - \theta_{Kinect}, d).$$

Floor and angle calculations are performed once every five frames.

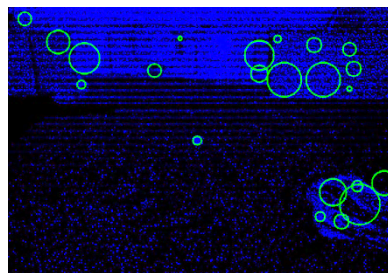
3.3 Robot Candidate Detection



(a) Raw Infrared Image



(b) Raw Depth Image



(c) Processed Depth Image

Figure 5

Navigation of Robots in Human Environments

After the ground plane and view angle are calculated, a set of all possible robot locations must be found in the image. Due to the distinctive shape of the Sphero, this consists of identifying circles in the image. Two strategies were explored in detecting the Sphero:

3.3.1 Luminescence-based Strategy

Because of the Sphero's glossy surface, the infrared image provides a greater contrast to the surroundings than the rgb-color image. The Sphero-detection algorithm works by finding differences in IR reflectivity—rather than IR intensity—in the input scene. As such, every input pixel in the IR image must be transformed to mitigate distortion caused by distance. As shown in Figure 5a, IR intensity is much greater closer to the Kinect and decreases as distance increases. The decrease in intensity was found to be linear with respect to distance. Every pixel in the input IR image is therefore transformed as follows:

$$I_F = 175(0.01dI_I)^5$$

where I_F is output intensity, I_I is input IR intensity, and d is distance to the particular point, in feet.

Due to the finite resolution of the Kinect's IR dot pattern, a series of speckles is clearly visible in the resulting image. A Gaussian blur with a size of 37 is applied to smooth out the image. A threshold filter is then run on the image to ignore dark regions. Finally, a Hough transform rates all potential circles in the image in order of circularity. Figure 5c depicts the Hough Transform results as green circles. Note the circled Sphero in the center of the image.

3.3.2 Depth-based Strategy

This method uses only the depth sensor to identify the Sphero, using a similar method to the one used to locate obstacle: Instead of determining Sphero location based on luminescence, the Sphero is differentiated by height from the floor plane. The 3-d world coordinates of each pixel in the

Navigation of Robots in Human Environments

depth-image are calculated. A pixel is set ON (1) if it lies above a particular height threshold—set as 3cm by default—or OFF otherwise. A Sphero in this image would resemble an oval—as only its top half is higher than 3cm above the ground. The image is therefore scaled in the y-direction upward by a factor of 150%. As with the luminescence based strategy, a Gaussian blur is applied to smooth out the resulting image, and a Hough transform is used to find circles in the image.

3.4 Sphero Detection

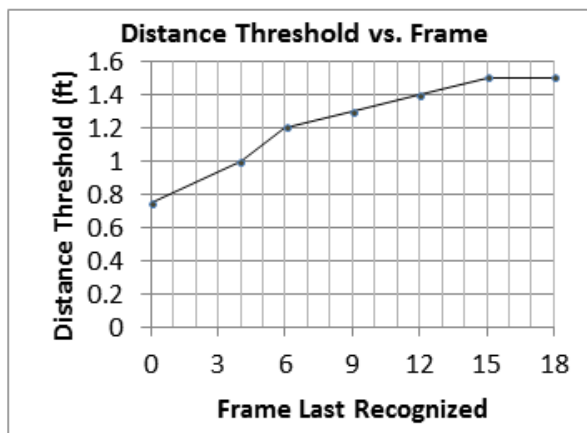
A 30-frame (≈ 5 second) long calibration step is performed upon initialization of the Kinect and Sphero Bluetooth connection. During these 30 frames, Sphero candidates are accumulated in an array. At the completion of the calibration step, a Sphero is assumed to be placed in the region of the image with the highest density of Sphero candidates over the calibration period. If no Sphero candidate persists for over 33% of frames in the calibration period, then the scene is assumed to contain no Spheros.

3.5 Removal of False Positives

A circle is removed if it is 2x bigger or 2x smaller than its expected size, given as a function of distance from the Kinect. Additionally, circles are removed if they map to regions with undefined depth or to regions greater than one foot above the floor plane.

3.6 Sphero Persistence

In order to track the Sphero’s movement in a particular frame, the location of the circle closest to the Sphero’s last known location is recorded. If the distance between the 3D world coordinates of the circle and the 3D position of the Sphero is less than a particular threshold, then



Graph 1

the Sphero’s position is updated to match the circle. The distance threshold is a function of the *time since the Sphero was last recognized*, modeled by Graph 1—under the assumption that the Sphero moves at around 1.0 feet per second, and moves at a relatively constant rate. A Kalman filter of position and velocity is used to account for noise in readings of the Sphero across frames. The Sphero’s position in 3-D space is provided to the filter, and velocity is calculated as the change in position divided by the elapsed time between subsequent time steps.

4. Obstacle Detection

The traversable world is described by a 2D 32x32 binary grid, each grid position spaced out 5.18cm apart—roughly 1.5cm larger than the radius of the Sphero (00cm)^[citation]. The Kinect’s depth image is analyzed in order to identify the set of impassable obstacles in the scene using the following algorithm:

Navigation of Robots in Human Environments

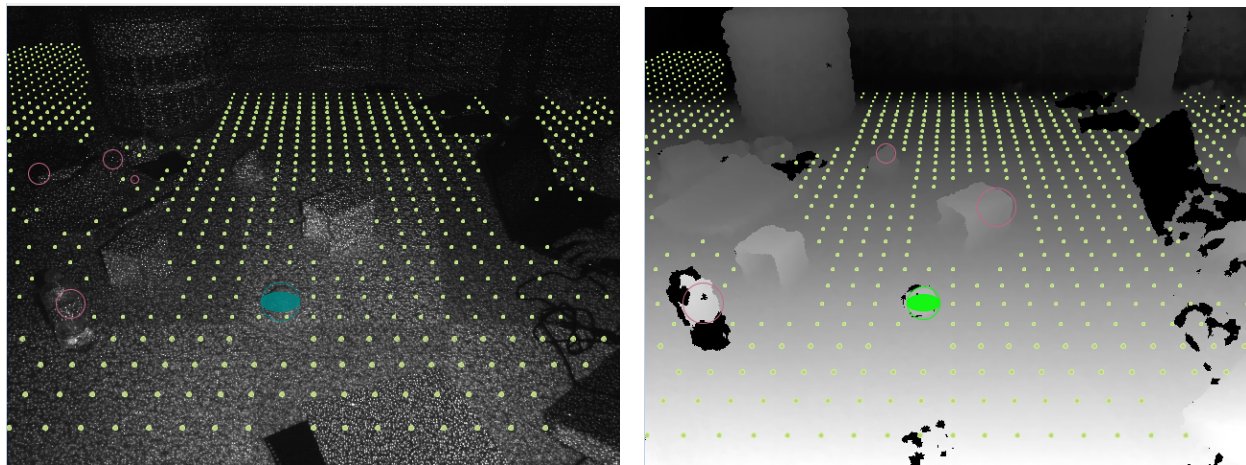


Figure 6a – Obstacle Map of a sample scene. (left) original IR image (right) original depth image. Dots represent traversable points on the grid. points is 10.16cm

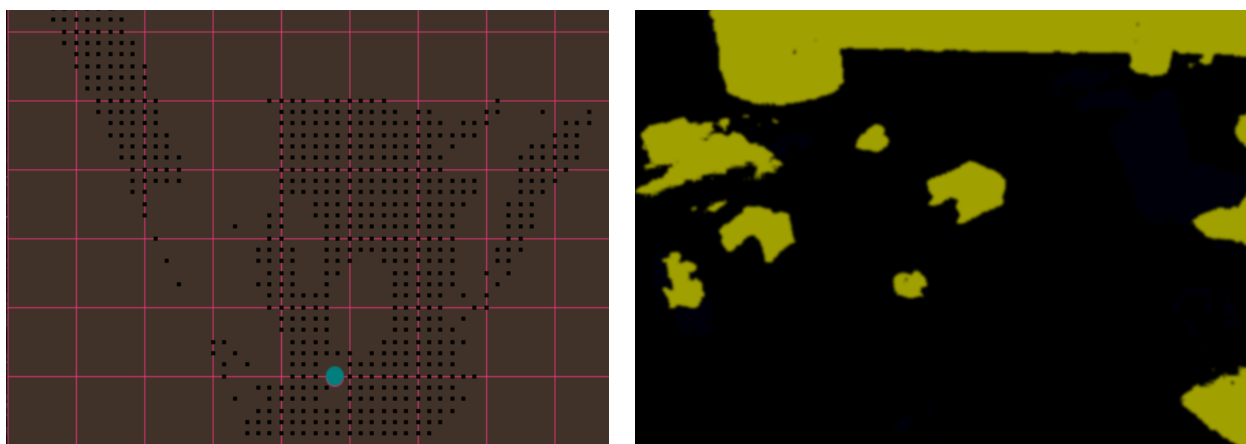


Figure 6b – Obstacle Map of a sample scene (right) Thresholded image: all points greater than 3.04cm above the ground (left) Resulting traversability grid. The z-axis extends positively upward, the x-axis extends positively to the right. The Robot is represented by the blue circle. Each cell in the grid has a size of 30.48 square cm. The distance between adjacent points is 10.16cm

For each pixel in the depth image, projected world position is calculated and stored as P . P_y persists in an array, and is updated as the weighted average between its last and current state; that is, if $P_y := 0.2P'_y + P_y$. If $P_y > 0.1 \text{ feet}$, then the pixel is defined as non-traversable. The weighted average is used to correct for random noise in input depth signal between frames. This analysis results in positions on this binary grid being marked as either traversable or non-traversable. The resulting image, Figure 6 (bottom right), represents the complete *obstacle map*. Next, the image must be transformed into a uniformly spaced 2-dimensional grid representing the world coordi-

nates of the floor. By definition, every point on this grid is contained within the floor plane. Each point is marked as traversable if both it and 8 surrounding points (roughly a Sphero-width away from the point) are marked traversable. The resulting traversability grid in 2 dimensions is displayed in Figure 6 (bottom left).

4.1 Pathfinding

Now that we have generated a 2D grid of all possible traversable locations, the A* algorithm is used to find the optimal path between two points, using an heuristic function that represents the diagonal distance between the start and end goal. The cost function between two nodes is modeled as

$$g(N_1) =$$

$$distance(N_1) +$$

$$\sum(\# \text{ adjacent nontraversable positions}),$$

in order to favor paths that do not follow too closely to walls. Diagonally spaced nodes are marked as traversable only if immediately adjacent tiles are traversable. Figure 7 displays the shortest calculated path between a start (blue) and end (top-right) node).

4.2 Line of Sight

As an optimization, periodic line-of-sight calculations are performed during the Sphero's path in order to reduce total travel distance and to remove redundant intermediate nodes. A path is drawn from the Sphero's current position to the position of the next node in its path. If this path does not intersect any impassable tiles, then the current goal is disregarded and the Sphero moves to the next goal in its path.

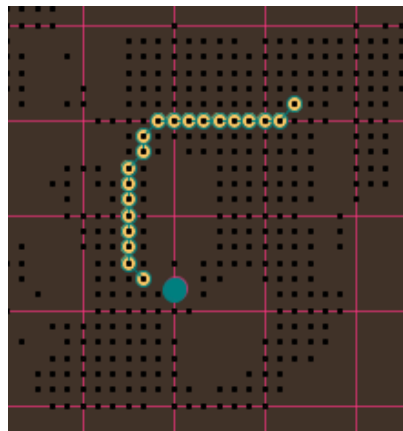


Figure 7 – Sample Calculated A* path. Black dots denote traversable nodes, the blue circle denotes the location of the Sphero, and the yellow dots illustrate the calculated path to a final destination. Note that the Sphero is blocked by too narrow a chokepoint on the right.

4.3 Path Changes

Approximately once every second, the robot's path is checked for integrity. Calculating if a previously-calculated A* path is still valid is a simple matter of visiting each of the nodes left in the Sphero's path, and checking if any of the nodes are now marked non-traversable. If so, then the path is recalculated. This system therefore smoothly handles the introduction of new obstacles—such as children's toys or wandering pets.

5. Motion Control

The current and next-target positions of the Sphero are now known. In order to move it towards its next target, however, it is important to know the Sphero's *current* bearing. The Sphero assumes a local bearing when it is turned on, and uses its built-in gyroscope and accelerometer to ensure that its bearing is always in the same global direction, regardless of how it has been rotated. A message to move forward at an angle of 0 degrees will therefore cause the Sphero to move forward in the direction of its local bearing. An input angle of 180 degrees will cause the Sphero to move in the opposite direction.

5.1 Calibration

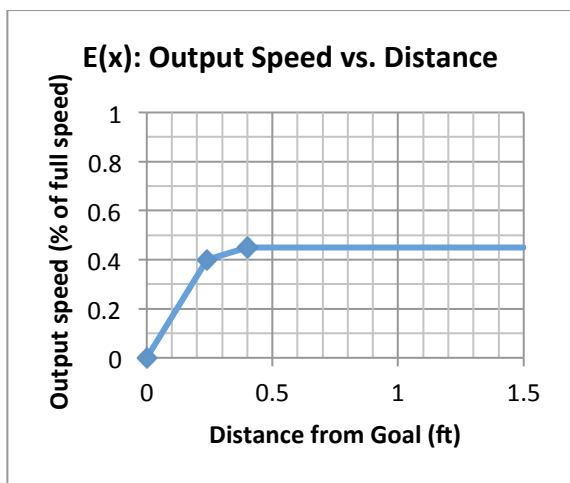
In order to calculate this angle, a calibration step occurs once the Sphero is found. A message is sent to the Sphero to move forward at a bearing of 0°. The *actual* angle of Sphero movement is calculated relative to the Kinect's direction vector: $\theta_{bearing} = atan2(ZOffset, XOffset)$. $\theta_{bearing}$ therefore represents the angle the line with a slope of (z/x), determined in the correct quadrant. Calibration occurs only once, at startup.

5.2 Bearing Computation

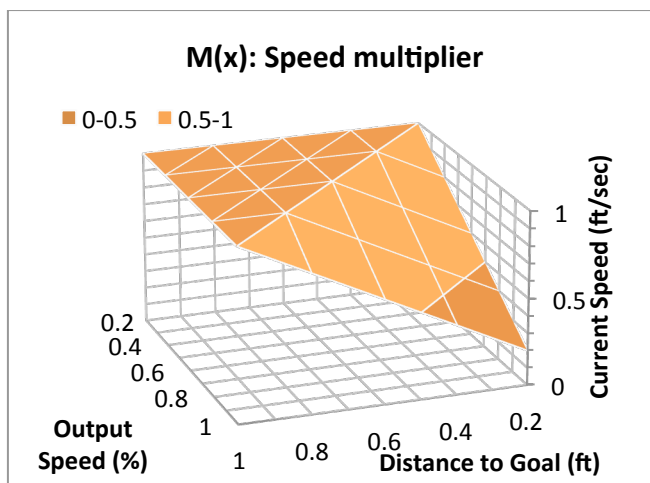
Once the Sphero’s local bearing is known, it is possible to orient the Sphero in the correct direction to travel to the next node:

$$\theta_{final} = \theta_{bearing} - \text{atan2}(\text{goalPosition.z} - \text{currentPosition.z}, \text{goalPosition.x} - \text{currentPosition.x})$$

Output speed is calculated as a function of distance to the next target and current speed: If $E(x)$ is modeled by the piecewise function in Graph 2, and $M(x) = \max [(1 + \text{distance} - \text{speed}), 1]$, then the speed output to the robot = $E(x) \cdot M(x)$. The multiplier reduces the possibility of the Sphero overshooting the target, by slowing down the Sphero if it is close to the target and speed is already high.



Graph 2



Graph 3

6. Results

6.1 Navigation:

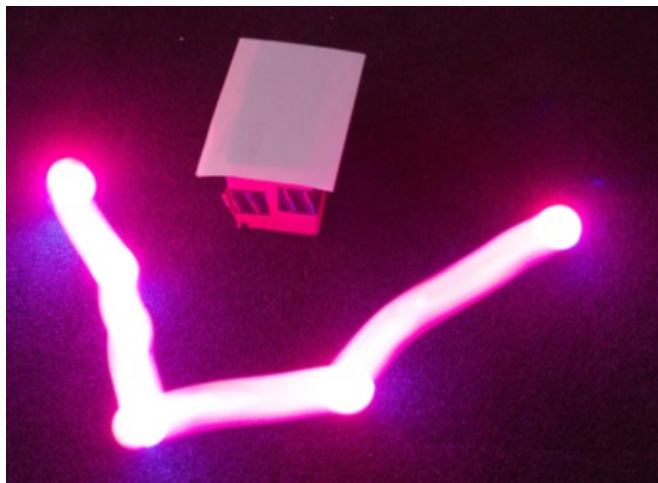


Figure 9



Figure 10

The system allowed the Sphero to be accurately navigated through a set of cluttered environments.

Figure 9 depicts a simple test case: the start and end goals were placed 90cm away from each other, with a $30 \times 30 \text{cm}^2$ box intersecting the path in the middle. Through 7 tests, the Sphero took an average of 8.9 ± 1.4 (std. dev.) seconds to complete the course.

Note that the distance between the path and box is a result of the angle of the Kinect. The region directly behind the box is not visible to the Kinect: the Kinect's angle and its lack of visibility into the region directly behind the box resulted in the computation of a wider turn radius for the Sphero.

The approach was successful even in more complicated setups. Figure 10 depicts the Sphero's path through a narrow corridor and around a sharp bend. The robot reached its targets with a variance of less than 0.2ft.

Navigation of Robots in Human Environments

Movement followed a stop-and-go pattern, as is evidenced by the white spots in the paths above. The Sphero would reach a point and pause for 1-2 seconds to rotate before continuing on to the next target

6.2 Sphero Detection:

The Sphero was reliably differentiated from the background, even in cases where the background and Sphero were similar in luminescence, and would have otherwise been indistinguishable in a color camera.



Figure 12 – The Sphero can be reliably located in cluttered environments.

The raw results from the Hough transform (Figure 11) contained a high number of false positives. However, the culling algorithms discussed section 3.5 reliably differentiated the Sphero from background objects. In the scene depicted in Figure 12, the Sphero was correctly identified 49 times out of 50 attempts.

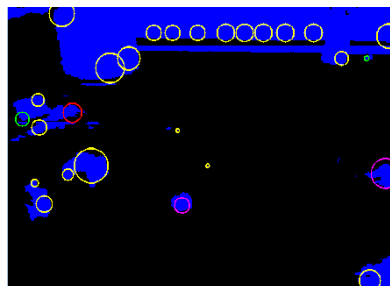


Figure 11 – Sample Hough Transform Results. This scene is the same as in Figure 6.

Recognition remained consistent during movement lateral to the view plane of the Kinect, though confidence decreased as the Sphero moved further away from the Kinect.

The depth strategy (see 3.3.2) worked very well when the Sphero remained within approximately 300cm from the camera. As distance increased, because of the Kinect's relative inaccuracy at higher distances, the Sphero became unable to be reliably differentiated from the environment. In

Navigation of Robots in Human Environments

environments with less reflective carpeting, however, the IR method of calculation excelled even at larger distances, because there is no effective limit on the IR camera.

Distance from Camera (cm)	Maximum Deviation (cm)	Maximum Deviation (% of Sphero Diameter)
90	0.2cm	3%
150	0.6cm	8%
210	2.0cm	27%
300	7.9cm	106%

Table 1

The Kinect's depth sensor was found to have a working range of approximately

400cm, after which the Kinect was unable to detect any geometry. Noise in measurements, however, tended to increase with distance (Table 1): a point approximately 90cm away was found to have a maximum deviation of only 2mm (0.08in—3% of the diameter of the Sphero) over 10 seconds of testing, while a point approximately 300cm from the Kinect a maximum deviation of 79mm (106%) over 10 seconds.

6.3 Algorithm Speed

The algorithm itself maintains a calculation time of approximately 100ms per frame, of which approximately 10-30ms are used in retrieving sensor data from the Kinect. RANSAC floor detection takes 15ms to execute every five frames. OpenCV image processing calls occur within 80ms, while Sphero control and localization take 1-2ms. Measurements were taken using Windows 7 (64-bit) on a dual-core Intel(R) Core(TM) i3-230M CPU @ 2.20GHZ with 4.00 GB of RAM, without any nonessential background processes running. Overall, the Sphero is located about twelve times per second: over 0.83 seconds, the Sphero can travel at a maximum of $\approx 3.8cm$ between frames—which provides adequate granularity to be found between frames.

6.4 Pathfinding

Paths must be calculated such that the Sphero will always have a large enough traversable distance between nearby obstacles: that is, at least a Sphero-length ($\approx 6cm$) must be allotted between each grid node. This effect is adequately achieved by allotting extra space on the near-side

Navigation of Robots in Human Environments

of obstacles and less between the far-edge of obstacles and the floor beyond. Better results may be achieved by mounting the depth sensor on a ceiling, pointing directly downward, or by using multiple sensors, viewing the scene at multiple angles.

7. Conclusion:

This research has provided insight into the feasibility of the use of an overhead camera to assist the navigation of small robots with very limited processing capability. No input from the Sphero itself was received or used for navigation. Thus, the external controller—connected to the Kinect—remained the sole method of both input and output, as would a stationary human controller. Overall, the system seems effective in navigating a small robot around a domestic environment. Very little time is spent analyzing the environment for obstacles, as the entirety of the room can be mapped out very quickly before the robot begins its path (see section 6.3). Provided the Kinect is placed at a favorable position (one where it can see the majority of the room), the robot can be very quickly and precisely navigated. Moving obstacles were also pre-emptively avoided in this overhead system.

In practice, it would seem that a more efficient system could be constructed with a combination of onboard and external techniques: for example, an odometer or small depth sensor may be placed on the robot itself for the purpose of local navigation in areas without direct coverage from the overhead system.

In the context of robots that must traverse the entirety of the ground surface area of a room, the overhead system offers several benefits by enabling efficient calculation of paths, recognition of unreachable areas, and self-diagnosis, should the robot entangle itself in the environment.

Navigation of Robots in Human Environments

While the Kinect works well in small areas, a higher-quality depth camera would be required for commercial use in larger rooms. The Kinect has a working range of only 13 feet (9 feet if the Kinect is tilted downward at a 45-degree angle), which is unsuitable for most rooms. Furthermore, the resolution of the depth image decays sharply with distance. Future research avenues the use of a larger, more recognizable robot than the Sphero; a more precise depth camera and synthesis of multiple depth cameras.

8. Definitions:

Robot	refers, in the context of this research, to this Sphero.
Sensor	refers to a depth sensor (eg. the Kinect)
Camera	see <i>Sensor</i> .
World Coordinates	Coordinate system centered at the position of the Kinect, with the vertical axis pointing in the upwards direction, normal to the ground plane.
Kinect Coordinates	Coordinate system centered at the position of the Kinect, with the z-axis pointing in the direction normal to the Kinect's view plane.
Screen Coordinates	2D coordinate system defined by the Kinect's input depth/IR image.

9. References

1. A Fischler, Martin A. and Bolles, Robert C. (1981). Random Sample Consensus: A Paradigm for Model fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24, 381-395.
2. Ackerman, Evan. (18 May 2010). BotJunkie Review: Neato Robotics XV-11. Retrieved 04 July 2012, from: <http://www.botjunkie.com/2010/05/18/botjunkie-review-neato-robotics-xv-11>.
3. GearFuse. (01 October 2010). Robotic Vacuum Paths Mapped and Compared with Long Exposure Photography. Retrieved 31 August 2013, from: <http://www.gearfuse.com/robotic-vacuum-paths-mapped-and-compared-with-long-exposure-pictures/>.
4. iRobot. What are Roomba® Virtual Walls®?. Retrieved 12 September 2013, from: <http://www.landmark.edu/library/citation-guides/landmark-college-citation-guides/apa-citation-style-guide/#Images>.
5. Layton, Julia. (03 November 2005). How Robotic Vacuums Work. Retrieved 01 September 2013, from: <http://electronics.howstuffworks.com/gadgets/home/robotic-vacuum.htm>.
6. Microsoft Corporation. (2012). Kinect Sensor. Retrieved 12 September 2013, from: <http://msdn.microsoft.com/en-us/library/hh438998.aspx>.
7. Microsoft Corporation. (Introduction to Kinect for Windows Audio) [Diagram], Retrieved 12 September 2013, from: URL (<http://www.microsoft.com/en-us/kinectforwindows/develop/tutorials.aspx>)