Adaptive Obstacle Detection and Avoidance on a Mobile Robot Using Depth Images

(Bachelor project)

Rik Smit, s1717928, H.smit.6@student.rug.nl,
Tijn van der Zant*

August 31, 2011

Abstract

One of the most important parts in mobile robotics is the navigation of the robot trough the environment. Essential for robot navigation is the detection and avoidance of obstacles in real-time. Many solutions to this problem have been found using all sorts of sensor devices and algorithms. This article will provide an implementation to achieve obstacle detection and avoidance using the Kinect sensor. Using an offline procedure, the obstacle detection module will be provided with essential information about the environment to make it run online at low computational effort. Obstacle avoidance will be achieved by potential field navigation using the detected obstacles.

1 Introduction

1.1 Robocup@Home team

The Robocup@Home team of the University of Groningen (Zant and al, 2010) strives toward creating a service and assistance robot for domestic environments (Wisspeintner, Zant, Iocchi, and Schiffer, 2000). Tasks of this robot include following people in a room, serving drinks and performing human-robot interaction. The focus of the team is in the creation of an adaptable software platform on top of an educational robot. One of the elements in the architecture is the detection of objects and obstacles. The detection of obstacles is of great importance to let the robot safely drive trough a room while avoiding objects coming into its path. Other useful applications benefiting from obstacle detecting are approaching objects to a close distance to pick them up, or the recognition of different objects. This paper will show the results of the implementation of an obstacle detector and will address some applications using the obstacle information.

1.2 Obstacle detection methods

In mobile robot navigation the detection and avoiding of obstacles is essential to let a robot safely move trough the environment. While the problem of detecting obstacles and avoiding them exists from the beginning of mobile robotics, there still is a lot research to do when it comes to real-time obstacle detection on mobile robots. Several solutions have been proposed in the form of sonar sensors, laser range finders and image processing. Many of successes have been made using these methods, all of them with their own advantages and disadvantages. Sonar sensors are cheap and easy to implement, but suffer from ghost echo's and inaccurate readings from different surfaces (Borenstein and Koren, 1988). Laser range finders produce a high level of detail but are expensive, as are stereo vision camera's which also require high processing power and sophisticated algorithms to extract obstacles. With the introduction of the Kinect sensor [1] and open source software libraries which allow us to access the Kinects capabilities from practically any software platform, a new cheap alternative arrives when it comes to reconstructing 3d scenes in real time.

*University of Groningen, department of Artificial Intelligence

[1] See www.xbox.com/kinect for more information on the Kinect

1.3 Calculating the depth image

Using active vision with an infra-red grid, the Kinect calculates depth values from observed distortions in the grid. As a result an 8-bit depth image is provided with the distance information. Each pixel in the image can be related to a point in the Kinect's view where the 8-bit value of the pixel provides an estimation of the actual distance from the Kinect sensor to that point. Previous approaches of calculating depth images include stereo vision cameras which calculate disparity maps (Gaspar, Santos-Victor, and Sentieiro, 1994) from which the (relative) distance values can be extracted. Many algorithms have been developed to perform this operation, all of them with their own advantages and disadvantages. In general the it is hard to calculate high resolution depth images in real-time on mobile robots using stereo vision.

1.4 Detecting obstacles

This paper aims to provide a method to detect obstacles from a depth image generated using a Kinect sensor. Since the Kinect still is a recent product and open source software library are only available since November 2010, not many results are available for vision using the Kinect on robot platforms. Performing obstacle detection on depth images has been proposed before with methods like with grouping and labelling (Jeong and Nedevschi, 2008). Other methods performed by mono vision use statistical methods (Jamal, Mishra, Rakshit, Singh, and Kumar, 2010). When it comes to detecting obstacles, besides the detection of objects on the ground, the ground itself is also detected as an obstacle. It is therefore necessary to subtract the ground plane from the rest of the scene. Jeong and Nedevschi (2008) uses a common approach by calculating the distance to the ground plane using the horopter [2] of the stereo vision camera's. This method is usable on constant surface conditions only and is therefore not usable on rough terrain. Jamal et al. (2010) uses a computational more difficult approach by statistically determining the ground plane from a image by looking at differences in RGB values. This method also requires constant values of the ground plane for both the structure as the material. Results show high distortions and weak resolution, which makes it unreliable in clumsy office environments. An other method of subtracting the ground plane is by calibrating an obstacle detector to a clear floor. The depth images retrieved from the Kinect provide useful information about the location of the floor with respect to the Kinect sensor. Using statistical methods, the floor can be subtracted from the rest of the scene in the image, leaving only the obstacles on top of the floor or gaps in the floor. The results of an implementation of an obstacle detector using this method will be shown in this paper.

[2] The volume centred on the fixation point that contains all points in space that yield single vision is called the horopter

1.5 Useful implementations

Several applications might use the information retrieved from the obstacle detector. This paper will show an implementation of an obstacle avoidance module integrated on the robot used by the Robo@CupHome team of the University of Groningen. The module uses the potential field navigation method (Khatib, 1985; Arkin, 1998). The repellent forces of obstacles are combined with the attractive force of the target. The resulting vector will be the final direction of the robot to head. This paper shows how the obstacle information can be converted to useful input for the avoidance module and how the vectors resulting from the forces are calculated. Also the paper will show how the obstacle detector can be used for other application like detecting objects on virtually any kind of flat surface like tables or a counter-top.

2 Method

Multiple methods have been found for detecting obstacles using different types of sensors. In mobile robotics, the use of sonar has been a popular approach from the beginning. Since sonar has it limitation when it comes to precision and range, solutions have been proposed using vision camera's with mono or stereo vision as well as laser range finders. The use of stereo vision or laser range finders overcome the limitations of 2D images, providing a 3D information about the scene. Next to the traditional (expensive) depth sensors, recently affordable depth sensors have come to the market,

using active infra-red vision to determine the distance to points in the range of view. An example is the Kinect sensor from Microsoft, which uses distortions in an infra-red grid caused by the surface's structure to extract depth information of the scene. The obstacle detection system described in this article uses the depth images provided by the Kinect sensor to locate obstacles on a mobile robot. The detected obstacles are used in the obstacle avoidance module of the robot for safe navigation trough the environment. The robot contains normal laptop modules [3], which means the detection and avoidance system should computationally not be to complex in order to let it run in real-time.

2.1 System Setup

For the system setup a Kinect sensor is used. The sensor provides a depth image at a rate of up to 30 frames per second, which allows for real-time obstacle detection. The goal is to use the obstacle detection module on a mobile robot platform. The sensor will therefore be placed on top of the robot Woody [figure 1], which is used for the RoboCup@Home competitions of 2011. As the sensor points downwards, it will have a clear view of the environment directly in front of the robot. The robot itself consists of a pioneer RX2 base, a commonly used robot in education. On top of the pioneer a frame is placed, containing multiple processing units and sensors for gaining information about the environment. One of these units will be dedicated to obstacle detection.

2.2 Ground plane determination

Ground plane determinations is the process of locating the ground plane in the depth image provided by the Kinect sensor. The ground plane can be defined as the places accessible by the robot. Once the location of the ground plane is determined, it can be subtracted from the rest of the image, leaving only the obstacles. Different methods have been found for locating a ground plane in an image. Methods like gradient estimation (Jeong and Nedevschi, 2008) and the RANSAC (Lacey, Pinitkarn, and Thacker, 2000) algorithm used on a Quadrocopter[4] [figure 2] deal with the process

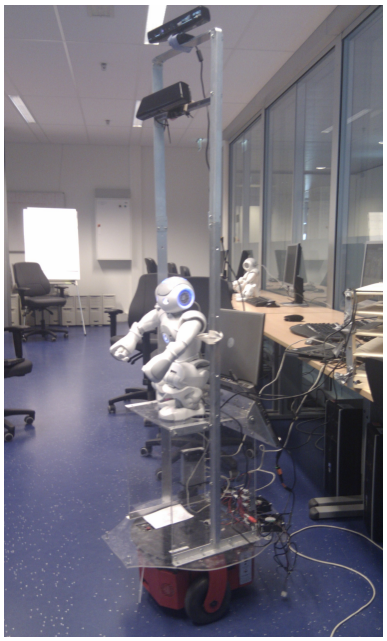

Figure 1: The robot used for testing the obstacle avoidance module. The Kinect sensor on top of the robot pointing downwards is used for obstacle detection. For movement, a Pioneer unit is used.

detecting and subtracting the ground plane in an online process. Both of these processes are computationally complex, and therefore other approaches have been found using an offline procedure for the complex task of determining the statistical properties of the ground plane (Jamal et al., 2010). Using this statistical information in the online procedure allows for fast subtraction of obstacles in images. An disadvantage of this last procedure has been the variability of light conditions and structure of the ground plane used in the online process, which makes the information found in the offline procedure unreliable. Since the depth images provided by the Kinect are constructed using active infrared vision, the structure and color of the ground plane, as well as the light conditions inside, have no influence on the retrieved depth data. The system even works in complete darkness.

The depth image contains depth values to every point in the range of view of the Kinect sensor up to a certain resolution[5]. The depth images provided

[3] Intel Pentium Dual Core 2.00Ghz laptops are used.

[4] http://www.youtube.com/watch?v=eWmVrfjDCyw

[5] The Kinect sensor provides a maximum resolution of

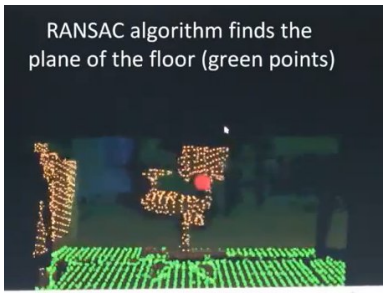

Figure 2: Results of an RANSAC implementation used on a quadrocopter to navigate trough a room. The yellow dots are point classified as obstacles, while the green dots are points classified as the floor.

by the Kinect sensor contain noise, meaning that not every pixel value in the image can be trusted as being a correct depth value. Most of this noise is the result of the method the Kinect uses to determine the depth values. To reduce the negative effects of this noise, the image will be divided into several segments containing n pixels each, resulting in a x by y grid. The averaging over the pixels p in each segment s_{xy} will be used as the depth value (equation 2.1) . Increasing the size of the segment will result increase the computational speed, but will decrease the precision in which obstacle can be detected. As a result 25 pixels per section is a good compromise. The average of each section is a good approximation of the depth value to the floor in that section.

$$s_{xy} = \frac{1}{n}\sum_{j=1}^{n} p_j^{xy} \tag{2.1}$$

The physical connection between the Kinect sensor and the rest of the robot is not perfectly rigid. As a result the sensor will sway as the robot moves. This means the relative location of the floor to the Kinect will change over time, which results in a variation of distance measurements while the location of the floor (and obstacles on the floor) remains constant. To overcome this problem multiple depth images, taken with different Kinect positions (as a result of movement), are used to calculate the ground plane values. The range within the minimum and maximum values of each segment s_{xy} over m images (equation 2.2), is considered to be the area in which the ground plane can appear.

640x480 pixels.

Algorithm 2.1 Ground plane determination algorithm

```
mins_xy ← 0 {Min pixel value over all images}
maxs_xy ← 0 {Max pixel value over all images}
for every image i do
  Divide image i in m x n segments s^i_mn.
  for every segment s^i_xy do
    Calculate average pixel value p in s^i_xy
    if p < mins_xy then
      mins_xy ← p
    end if
    if p > maxs_xy then
      maxs_xy ← p
    end if
  end for
end for
```

$$s_{xy} = min/max_{k\in m}\left(s_{xy}^{k}\right) \tag{2.2}$$

The complete algorithm is described in algorithm 2.1

2.3 Obstacle subtraction

In the real-time obstacle detection procedure, from every retrieved depth image a segment grid is created in the same way as used for determining the ground plane (equation 2.1). Obstacle detection is achieved by comparing the segment grid to the ground plane data collected in the offline procedure. If the depth value in a specific segment s_{xy}^{c} significantly differs from the depth value of the corresponding segment in the ground plane s_{xy}^{i}, either an obstacle (equation 2.3) or a gap (equation 2.4) is at that position [figure 3].

$$s_{xy}^{c} < \left(s_{xy}^{i} - e\right) \rightarrow Obstacle \tag{2.3}$$

$$s_{xy}^{c} > \left(s_{xy}^{i} - e\right) \rightarrow Gap \tag{2.4}$$

The error e specifies an extra threshold value that the segment value had to pass. The error can be adjusted to allow the robot not to detect obstacles or gaps with a certain height. This can be used to ignore for instance a doorstep or small elevations in the ground plane, which should not be seen as obstacles to the robot as the robot can easily drive

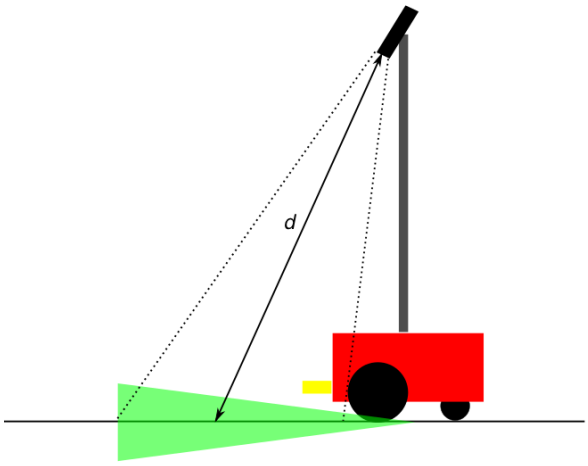

Figure 3: The Kinect sensor on top of the robot will look down to detect obstacles in front of the robot within the range of view. The distance *d* from the sensor to points on the floor will be used for ground plane subtraction. Because the sensor will sway as the robot moves, distance information within a certain range (green) is used for reliable results.

Algorithm 2.2 Obstacle substraction algorithm

while Robot running **do**
  Retrieve image i from Kinect
  Divide image i in m x n segments s^i_{mn}.
  for every segment s^i_{xy} **do**
    Calculate the average pixel value p in s^i_{xy}
    if $p < mins_{xy} - e$ **then**
      There is an obstacle at this position
    end if
    if $p > maxs_{xy} + e$ **then**
      There is an gap at this position
    end if
  end for
end while

over them. See algorithm 2.2 for the obstacle subtraction procedure.

2.4 Obstacle avoidance

The detection of obstacles on a mobile robot has numerous applications. One of the application that always has been going hand in hand with obstacle detection is obstacle avoidance. In this section an implementation of an obstacle avoidance module is given to show the potentials of the obstacle detection module. The implementation uses potential field navigation, described in many papers and books like (Khatib, 1985) and (Arkin, 1998). Potential field navigation uses the location of detected obstacles to create a potential field, used by the robot to navigate trough the environment.

The obstacle detection module provides a grid with all obstacles in front of the robot, seen from above. The avoidance module will respond to these obstacles to make the robot move away from them. For useful obstacle avoidance, the module has to decide which of the found obstacles are relevant to the robot. In general, the obstacles closest to the robot should be weighted more relevant than obstacles further away. In the current setup however, not all obstacles close to the robot are relevant. Instead obstacles that are not directly reachable from the robot's point of few do not need to be taken into consideration when using the potential field navigation method. One way to think of this is as if the robot is observing the environment in front of it by using sonar. algorithm 2.3 shows an implementation to convert a normal obstacle grid as provided by the obstacle detector to a new obstacle grid that is more relevant to use with potential field navigation. Figure 5 shows an example of the result of the algorithm. All the red squares are obstacles found by the obstacle detector. The filled squares are the obstacles selected by the algorithm. The algorithm will scan the obstacle image like an radar from the robot's point of view, to subtract the useful obstacles. The algorithm will start looking at -90 degrees (left) and increase the angle to 90 degrees (right). At every angle, the algorithm will look further away by increasing a (virtual) line, until the end of the line touches an obstacle. At the moment an obstacle is found, or no obstacle is found at all, the angle will be increased to find the next obstacle.

For every obstacle in the range of the robot, a

Algorithm 2.3 Find useful obstacles in an obstacle grid

```
for every angle a do
  Line length l ← 0 {Start looking closest to the robot}
  while Line length l <maximum length OR no obstacle is found yet do
    increase l
    if l reaches an obstacle o then
      add o to found obstacles
    end if
  end while
end for
```

vector is calculated. The vector describes the influence the obstacle has on the robot. The vector points from the obstacle to the robot to indicate the robot should avoid the obstacle. The length of the vector determines the amount of influence the obstacle should have on the robot. The closer the obstacle, the more influence the obstacle has, as it becomes more important to avoid the obstacle. The calculation of the vector length as function the the (normalized) distance to an obstacle d is the key to a successful obstacle avoidance behaviour with this implementation. This module will use an exponential function to give obstacles closer to the robot exponentially more influence, as done by Khatib (1985) (equation 2.5). The exponential allows obstacles further away to have almost no influence as the probability of a collision with them is very small, but obstacles very close to have very much influence, as it is very important to avoid these obstacles immediately. The difference with a normal linear function is that the exponential function will result in a more natural behaviour.

$$length = \left(\frac{1}{d} - 1\right)^2 \tag{2.5}$$

By combining all the vectors generated from the obstacles, one final obstacle vector is generated. The final obstacle vector will be formed by taking the average angle and the average length of each obstacle vector. The resulting vector describes the common rejective influence that the obstacles have on the robot. Next to the obstacle vector, a target vector is provided. This target vector points in the direction the robot wants to go. The length of the target vector will depend on the distance to the point the robot wants to go to, which will be used to decrease the speed of the robot when the target goal becomes closer. The resulting behaviour will evenly slow down the robot, instead of performing a hard stop when the robot arrived at the target location. Combining the final obstacle vector with the target vector results in a final vector, corresponding to the final direction the robot should go. This direction will correspond to a movement away from the obstacles, as well as a movement towards the target goal. When the avoidance module is tuned well enough, the robot will never collide with obstacles under normal (unexpected) conditions.

3 Results

This section will mainly focus on the results of the obstacle detection module. Since the detection and the avoidance module are going to be used on the robot for the RoboCup@Home competitions, the performance of the modules on this robot will be kept in mind. In order to achieve good performance, the modules should be able to run in real-time on a normal household laptop (Intel Dual Core 2Ghz) as used on the robot.

3.1 Obstacle Detection

The performance of the obstacle detection module greatly depends on the settings used. Adjusting the size of the segments in the grid used to detect the floor will influence both the precision as well as the computational speed of the obstacle detection. The settings used to measure the performance for our purposes are based on the input format of the depth image (640 by 480 pixel grey-scale image with 8-bit depth). Using segments of 5x5 pixels proved to be a decent compromise. The size of the segments can be determined on the fact that they should be large enough to handle the noise that comes from the depth sensor. The noise normally consists of a few pixels with values significantly different from adjecent pixels, while the distance is similar. Although the depth images provided by the depth sensor in general don't consist a lot of noise, even small amounts can result into false obstacle detection if not handles carefully. The segment size of 5 by 5 pixels will cover the noise, as potential outly-

ing pixels will be flattened by surrounding normal pixels. Enlarging the segment size will increasingly lower the noise, however it will have an impact on the performance when it comes to detection precision. Only one obstacle can be detected per segment, which means the detection resolution (number of possible detection per image) as well as the detection precision (how large an obstacle need to be before it can be detected) will decrease.

The time needed for retrieving the depth images to be used in the obstacle detection module depends both on the operating speed of the Kinect sensor itself and the software libraries used for converting the data of the Kinect to useful depth images. The overall speed for retrieving depth images is approximately 30 frames per second. For obstacle detection on a mobile robot in real-time, this should be considered fast enough. The performance of the obstacle detection module therefore depends on the speed of detecting obstacle in the depth images. The algorithm used for calculating the ground plane data requires approximately 2.1e-5 seconds per frame. Since this is an offline procedure, the algorithm does not necessary need to perform in realtime speed, because the calculation of the ground plane is done before the robot is put to operation. The current results however show that realtime ground plane calculation is no problem for the algorithm. The algorithm used for obstacle detection in the online procedure takes about 1.4e-6 seconds per frame. These numbers show that the main limitation for obstacle detection is the calculation of the depth images. Using these settings allow the obstacle detector to run in real-time at about 30 frames per second.

The size that obstacles should have to be distinguished from the floor is an import performance measurement for the obstacle detector. When used on a mobile robot for obstacle avoidance purposes, the detector should at least be able to detect obstacle of the size it should avoid (i.e. cannot drive over). The setup used in this research lead to the data shown in table 1. The results correspond to a detection range of approximately 4-6 cm when the robot stands still, and 4-7 cm when the robot is moving. The lower and upper bound of these measurements are the result of the variety of location of obstacles in the depth image. Obstacles further away from the robot are detected with less precision, since there is less coverage of pixels per centimetre of floor size. This means that in order to be sure that an obstacle is detected on a moving robot, it should at least be 7 centimetres in height. The difference in detection range between a robot standing still and a robot moving is the result of the movement of the Kinect sensor on the robot. When the robot is moving, the sensor will also move (back and forward) which leads to less reliable depth data.

Table 1: Results of the obstacle detection method. The data should give an idea about the variety in distance values retrieved from the depth image. The smaller the deviation, the higher the precision of the detector.

	steady	moving
Average distance value	57.4 62.9	58.0 63.2
Minimum distance value	57.3 62.8	57.1 62.5
Maximum distance value	57.5 63.1	58.2 63.5
Mean minimum deviation	**0.24%**	**0.68%**
Mean maximum deviation	**0.27%**	**0.84%**

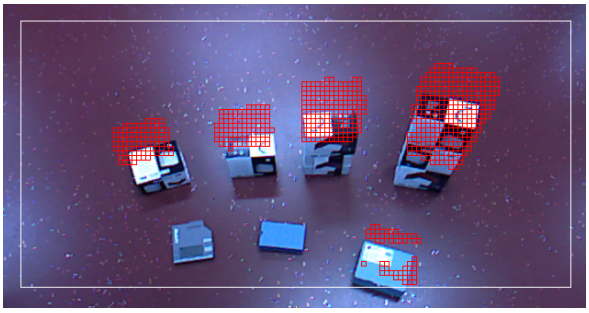

Figure 4: The red squares represent the segments classified as an obstacle. The location of the obstacles are biassed upwards as a result of using the RGB image provided by the Kinect as the background image instead of the depth image, which is actually used for detecting the obstacles. The image shows the limitations of the obstacle detecting module, as not all the objects in front of the robot are classified as an obstacle.

3.2 Obstacle Avoidance

The best way to test the obstacle avoidance module is by using it in real-time situations on a mobile robot. Challenges for obstacle avoiding using potential field navigation have always been small corridors the robot has to go trough, and dead-

ends where the robot should move away from. Also should the robot be able to deal with moving objects like people passing in front of the robot. The problem of not being able to go trough a small corridor (just wider than the robot itself) is a well known problem in potential field navigation, since the robot will react on both the left and the right side of the corridor. The resulting goal direction might be turning backward, since this is one way to avoid the walls. In general, much tuning is needed to adjust an avoidance module using potential field navigation so that it will go trough the corridor (avoiding both walls) instead of turning back (also avoiding both walls). The other well known problem of being stuck at a dead-end appears when the robot is surrounded with obstacles or the robot is facing a wall right in front of it. Since both to the left as the right of the robot are obstacles (or a wall), the robot has a problem to decide which way to go, resulting in endless nervous shaking to the left and right instead of turning all the way. It is a challenge to avoid this behaviour without biassing the robot to a certain direction, or manually adding noise. The avoidance module has been tested in the robot-lab of the Artificial Intelligence department of the University. The lab contains obstacles like chairs, boxes and tables (all within the minimum height required by the obstacle detector). Several 10 minute runs show that the obstacle avoider guides the robot trough the lab without touching obstacles at a maximum speed of around 0.9 m/s. Also people passing the robot result into no problems, as the robot sees them as obstacles and will continuously try to avoid them. The robot is capable of driving trough narrow corridors and doorways, as well as avoiding gaps that are to small to pass. When encountering a dead-end in a corridor for example, the robot will eventually (after about 3 seconds) turn around and drive away by itself instead of continuously 'hesitating' whether to turn left or right. This behaviour is not hard-coded into the avoidance module, and is therefore a positive side effect of the normal behaviour. The main limitation of the avoidance module is the avoidance of small obstacles, which are not detected by the obstacle detector. Figure 5 shows an visualization of the obstacle avoidance in progress from the robot's view. In this case the robot will avoid the computer case by driving around it to the right.

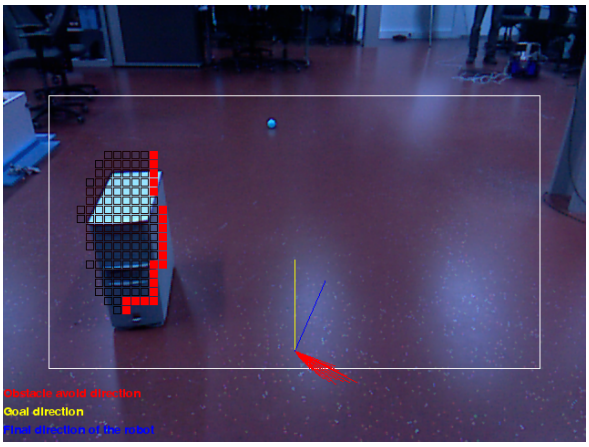

Figure 5: From all the detected obstacles (red squares) only the the obstacles in view of the robot (filled squares) are used. The red vectors show the repelling force the robot experiences from the obstacles. The yellow vector is the command the robot is given ('move straight forward'). The blue vector is the final direction the robot will go, as a result of combining the red vectors with the yellow vector.

4 Discussion

In this paper I proposed a fast and robust way to detect obstacles on a mobile robot. The results show an obstacle detection algorithm that is capable of processing an depth image at around 1.4e-6 seconds on a standard laptop. The bottleneck of the obstacle detection module is therefore the speed at which the depth images are retrieved from the sensor. For the Kinect sensor this is around 30 frames per second. When it comes to the precision, the main bottleneck also is the resolution of the data provided by the Kinect. Using more precise depth images will directly improve the performance of the obstacle detector. Next to the obstacle detection, an obstacle avoidance module is proposed to show an useful implementation of the obstacle detection algorithm. The results show that the robot is capable of performing real-time obstacle avoidance under uncertain dynamic conditions. The limitations with the setup used in this paper can be found at the detection of small objects up to around 4 to 7 centimetres. Further work on the detection module can be the implementation of an method to do find the ground plane data online, instead of offline as

described in this paper. This will allow for more adaptive obstacle detection on surfaces of different heights. Also, as this paper focusses usability of the implementation and not on optimization, work can be done to test the optimal settings in different conditions to optimize the performance of the modules in specific situations.

References

R. C. Arkin. *Behavior-Based Robotics (Intelligent Robotics and Autonomous Agents)*. MIT Press, 1998.

J. Borenstein and Y. Koren. Obstacle avoidance with ultrasonic sensors. *IEEE Journal of Robotics and Automation*, 4, 1988.

J. A. Gaspar, J. Santos-Victor, and J. Sentieiro. Ground plane obstacle detection with a stereo vision system. *International Workshop on Intelligent Robotic Systems - IRS94*, 1994.

A. Jamal, P. Mishra, S. Rakshit, A. K. Singh, and M. Kumar. Real-time ground plane segmentation an obstacle detection for mobile robot navigation. *Emerging Trends in Robotics and Communication Technologies (INTERACT), 2010 International Conference on*, 2010.

P. Jeong and S Nedevschi. Obstacle detection based on the hybrid road plane under the weak calibration conditions. *IEEE Intelligent Vehicles Symposium*, 2008.

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, 2:500 – 505, 1985.

A. J. Lacey, N. Pinitkarn, and N. A. Thacker. An evaluation of the performance of ransac algorithms for stereo camera calibration. *British Machine Vision Conference*, pages 646 – 655, 2000.

T. Wisspeintner, T vd Zant, L. Iocchi, and S. Schiffer. An evaluation of the performance of ransac algorithms for stereo camera calibration. *British Machine Vision Conference*, pages 646 – 655, 2000.

T vd Zant and al. Borg - the robocup@home team of the university of groningen team description paper. Master's thesis, University of Groningen, Groningen, 2010.