Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)

STC Robot

Optimally Covering an Unknown Indoor Environment

Project by:
Majd Srour
Anis Abboud
Under the supervision of:
Yotam Elor and Prof. Alfred Bruckstein
and technical support by:
Sergey Danielian

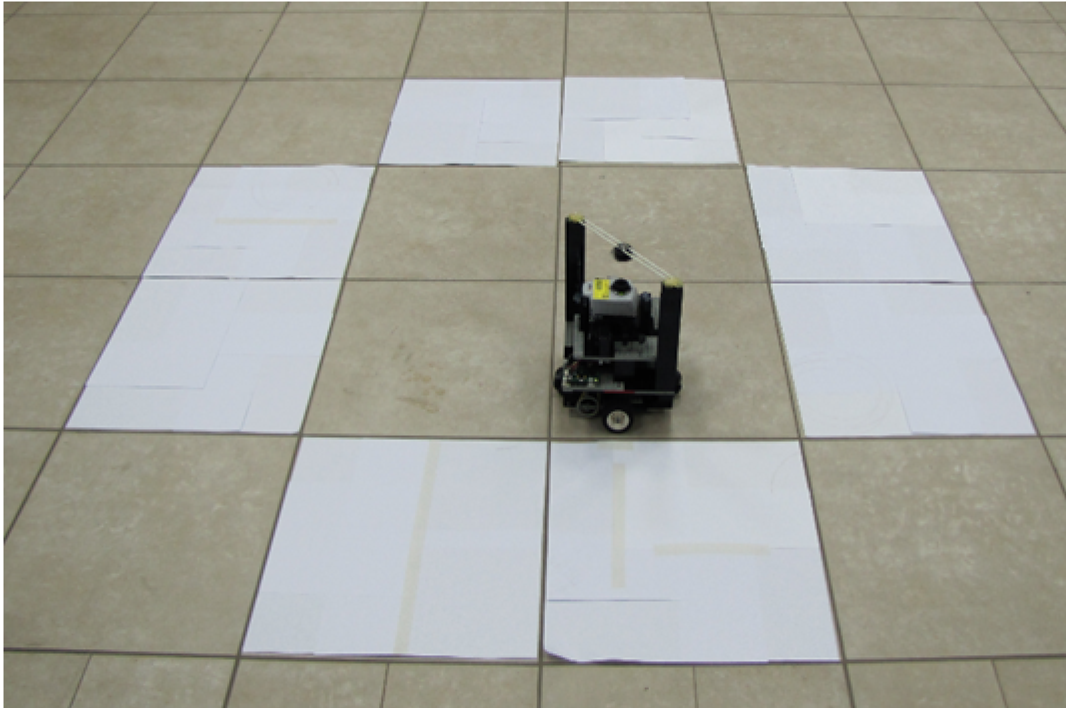
Table of Contents

- STC Robot
- Table of Contents
  - Task Description
  - The Robot
  - The Transformation
  - The Algorithm
    - On-line STC algorithm
    - Example
  - Implementing the Algorithm
  - Angle Detection
  - Obstacles Detection
  - Connectivity Issues
  - Videos

Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)

Task Description

Given a floor like this:



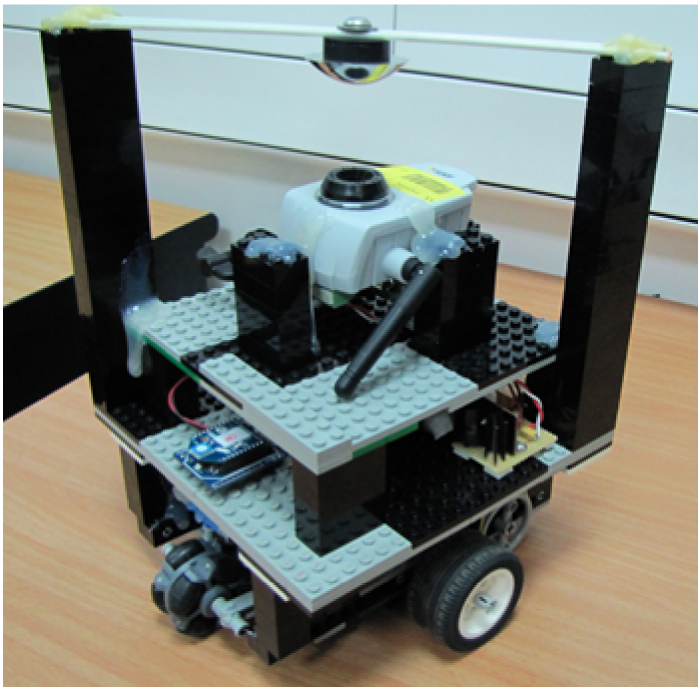
Where the gray cells are the free cells, and white cells describe obstacles.
Each gray cell is divided into 4 *sub-cells* each of the same size as the robot.
The Robot's task is to cover the floor, passing exactly once over each *sub-cell* and to return to the same cell where it started.

Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)

The Robot

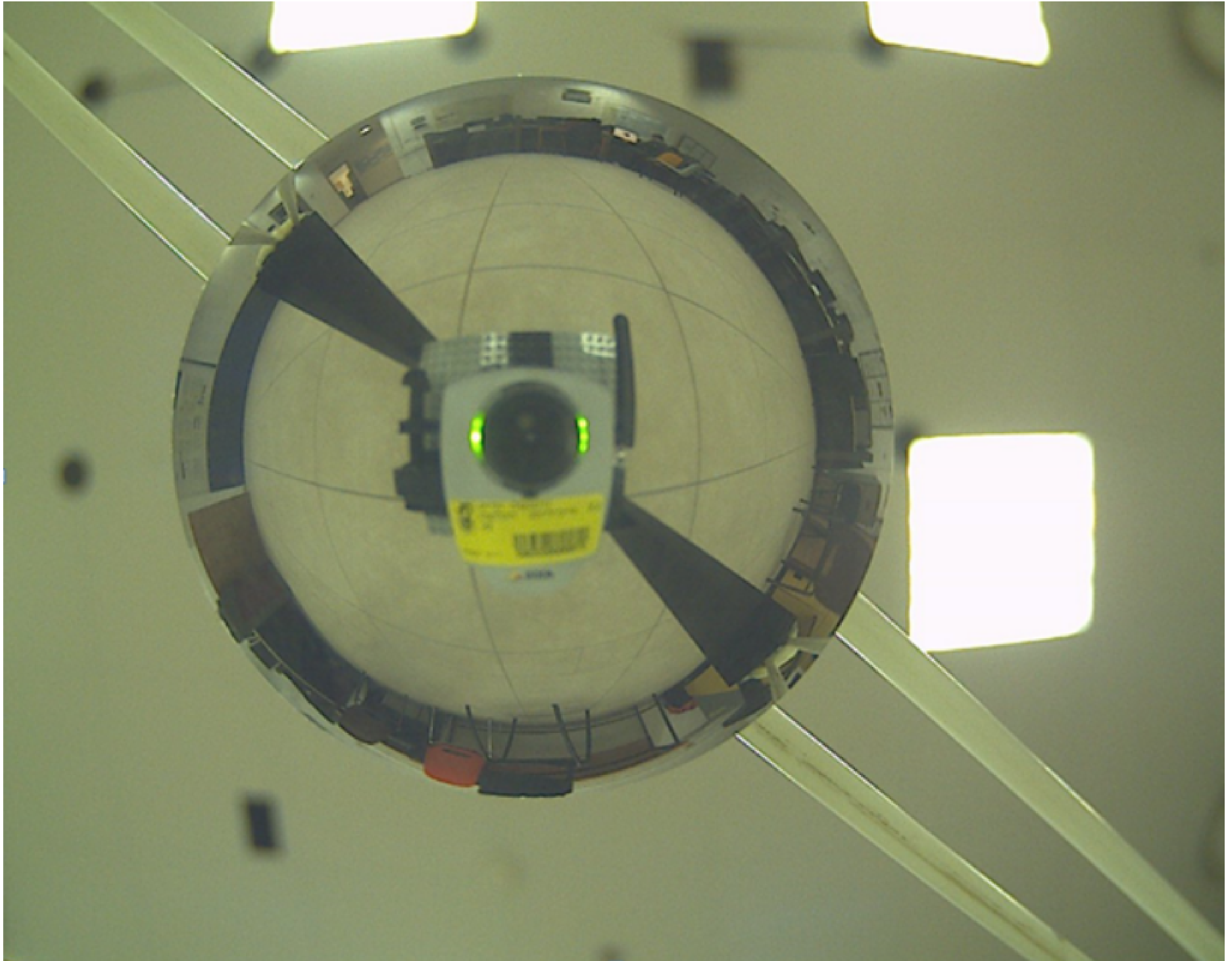
The Robot (figure 2) is built from *Mindstorms* Lego. Its major parts are:

- **Board:** *PIC-18* board.
- **Wireless Module:** *Xbee* wireless communication module.
- **Wheels and Motors:** 2 main wheels, motorized by two gear motors, and two auxiliary wheels that enable the Robot to perform an on-place 90 degree rotation.
- **Camera:** Wireless camera directed up.
- **Mirror:** A curved spherical mirror, directed down.



The Robot is wireless-controlled by a computer program which runs the *STC Algorithm.*
The camera with the spherical mirror, enable the Robot to take a 360 degree panorama view of the adjacent area in a single shot, as shown below (figure 3).

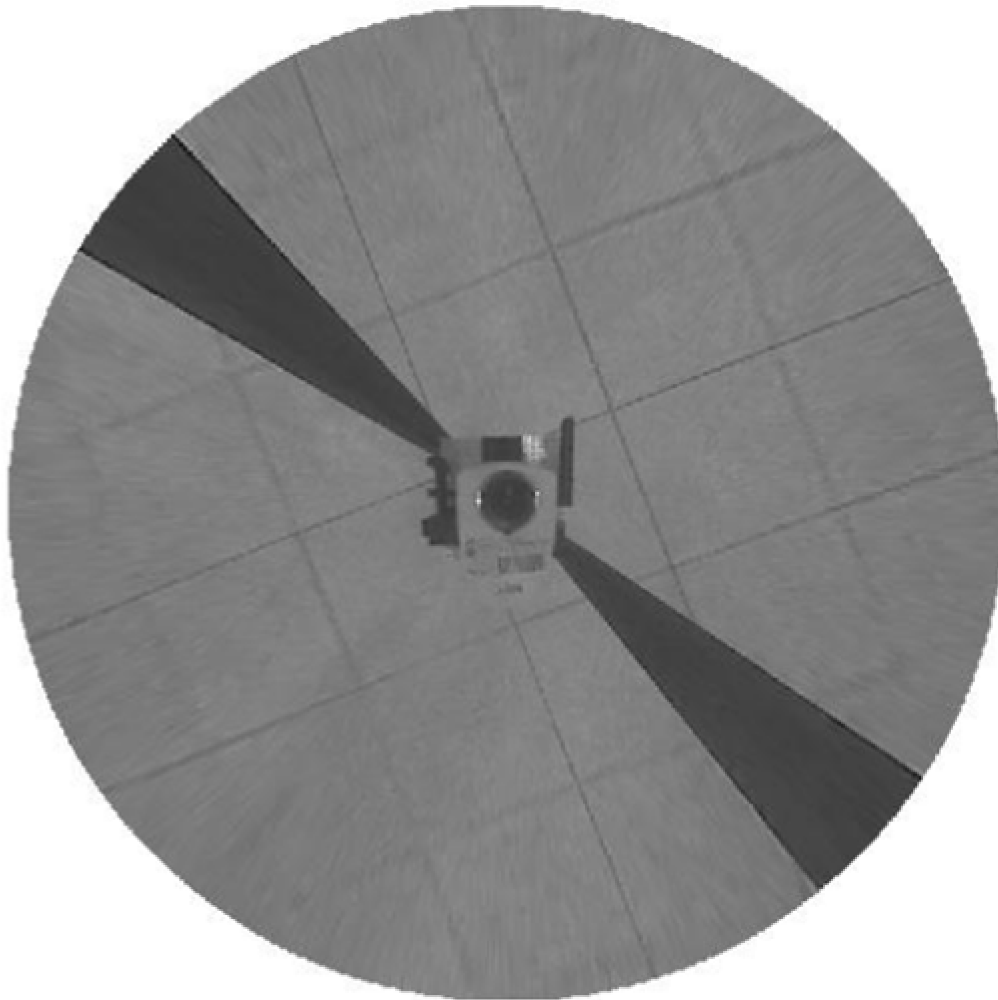
Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)



Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)

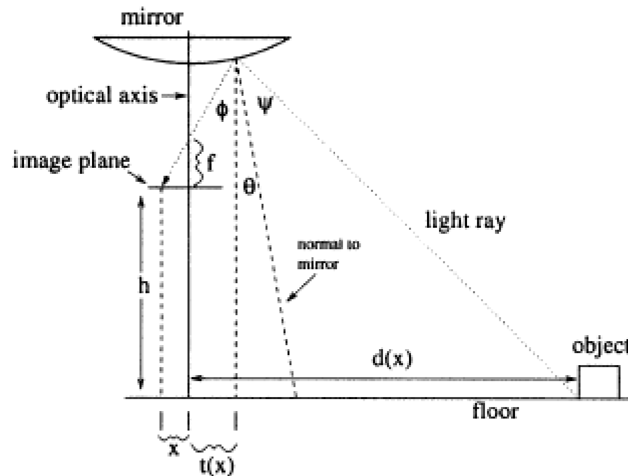
The Transformation

In order to work with the 360 degree panorama view above, we apply a transformation, which produces an image like the one shown in figure 4.



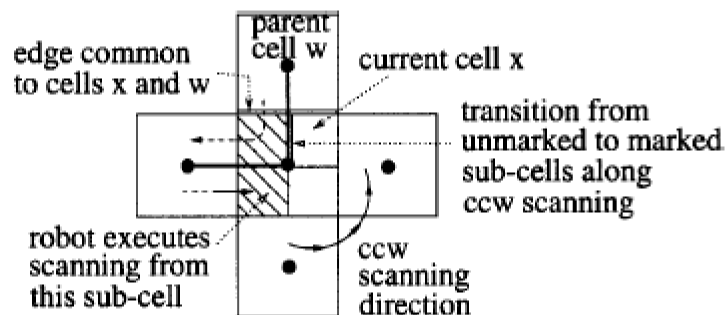
The main idea of the transformation is that if we look at a single pixel in the *curved* image, and compare it to its new coordinates in the *normal transformed* image, we see that the pixel moves on the axis that connects it with the camera lens. In other words, in both the *curved* picture and the *normal* picture, the pixel is in this axis although in the curved picture the pixel is closer to the camera. So what we need to do is to calculate the ratio between its distances of the camera lens in the two pictures. How much should it move on this axis in order to get the *normal* picture, we did this by calculating the angle *Phi* and *Theta* as shown in figure 5.

Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)



The Algorithm

The Robot uses the on-line version of the STC Algorithm described in the paper "Spanning-tree based coverage of continuous areas by a mobile robot" by Yoav Gabriely and Elon Rimon (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf). A copy of the paper is attached in the appendix.
We assume that the robot occupies a square of size D.
We subdivide the work-area into square cells of size 2D, and assume that obstacles cover complete cells only.
In this version of STC the robot has no prior knowledge about the environment, except that obstacles are stationary. Rather, the robot must use its on-board camera to detect obstacles and plan its covering path accordingly. We assume that the robot is capable of identifying obstacles in the four cells neighboring the robot's current cell, as described in the following figure:



On-line STC algorithm

Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)

Initialization: Call STC(Null, S), where S is the starting cell.
STC(w, x): (A recursive function, x is the current cell and w is the previous cell along the spanning tree)

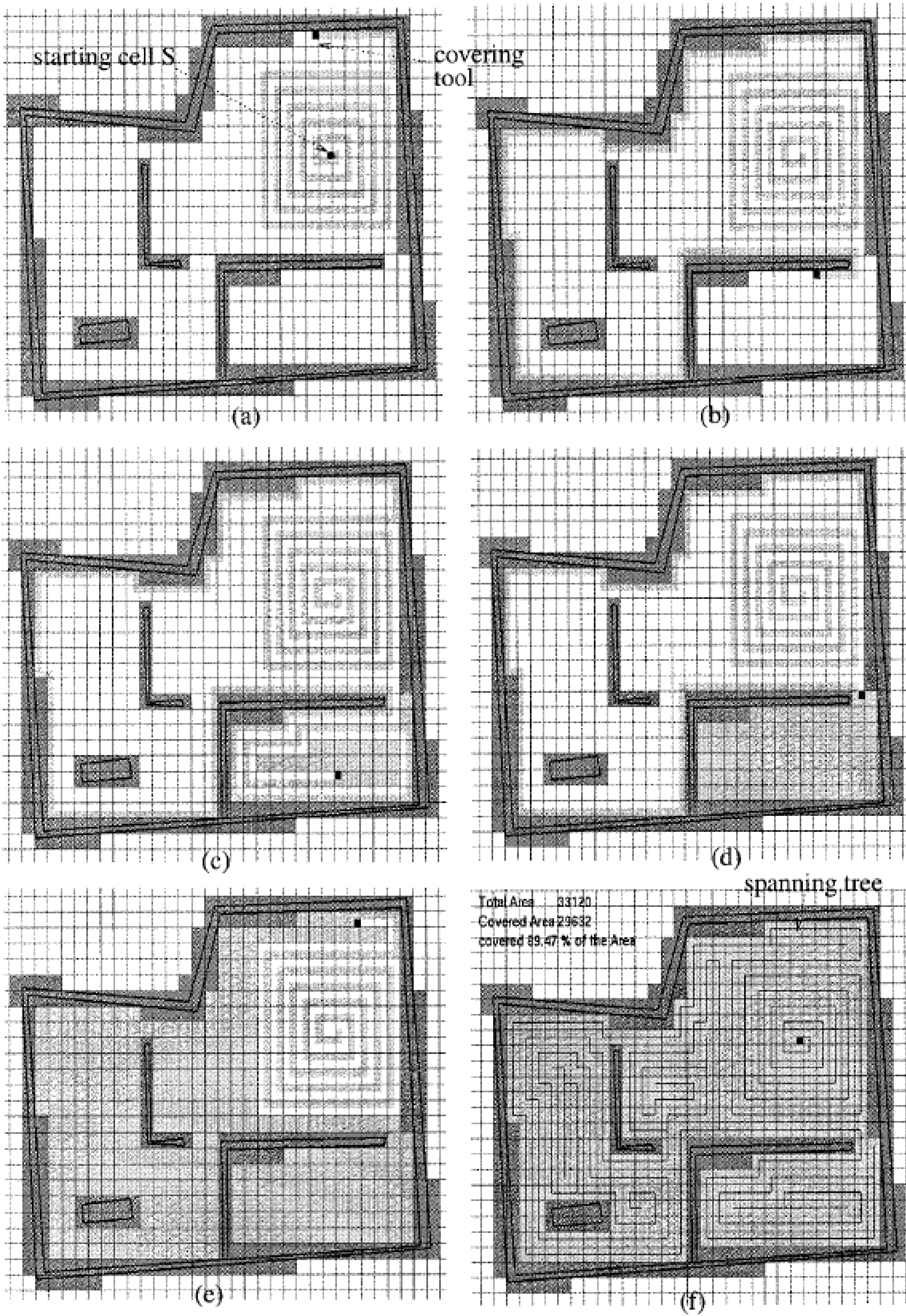
1. Mark the current cell x as an **old** cell.
2. While x has a **new** obstacle-free neighbor:
  - **2.1.** Scan for first **new** neighbor of x in counterclockwise order, starting with parent cell w. Call this neighbor y.
  - **2.2.** Construct a spanning-tree edge from x to y.
  - **2.3.** Move to a sub-cell of y as described below. The robot moves from its current sub-cell in x to a sub-cell of y by following the right-side of the spanning tree edges, measured with respect to the tool's direction of motion, as described in the figure above.
  - **2.4.** Execute STC(x, y).

  End of while loop.
3. If $x \neq S$, move back from x to a sub-cell of w as described below. When the covering tool returns to a parent cell w, it again moves through sub-cells that lie on the right-side of the spanning-tree edge connecting x with w, as shown in the figure above.
4. Return. (End of STC(w, x).)

Example

Link to this doc: http://goo.gl/kVkbV

STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)



(The figure above is taken from the STC Algorithm paper).

Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)

Implementing the Algorithm

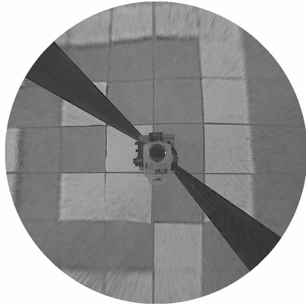
We save the board as a two-dimensional matrix of *sub-cells*.
Since the robot has no prior knowledge about the environment, we set the initial size of the matrix to 18×18 (9×9 *cells*). The robot will be placed in the bottom-right *sub-cell* of the central *cell* (matrix[10][10]).
Each time the robot reaches an edge of the matrix, we re-allocate the matrix with new_board_size = board_size * 2 - 2, putting the old data in the center of the larger matrix.
In general, the matrix dimensions are $2^n + 2$.
We also save a two-dimensional matrix of *cells*, which saves the parent *cells*.

Angle Detection

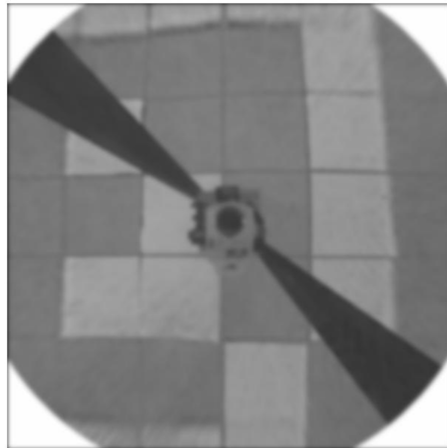
In order to detect the robot's deviation of the desired path, we should calculate the slope of the black lines between the ground cells.
We first examine the pixel-matrix of the image and calculate the gradient for each point.
After that, we look for the linear lines with the highest gradient in the matrix. That's where the black lines between cells are. We calculate the slope of these lines.
This is because on the edges of these lines, the derivative will be maximal, moving from black to gray-white pixels. We calculate the angle between the *x-axis* gradient, and the *y-axis* gradient using *atan2* function. We then create a *histogram* of those angles, the angle with the maximum value is the angle the Robot is rotated in. For this to work correctly, we needed to hide the camera with a black rectangle and the two poles holding the mirror with black lines so that they do not affect the calculations. Also, because of the curved mirror there is a lot of noise in the picture, and the calculation of the gradient alone was not accurate enough, so we needed to *smooth* the picture with *convolution function.*

Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)

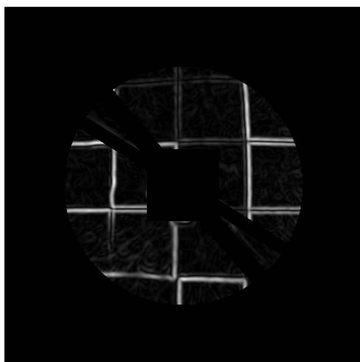
More *Technical* details about the Angle Detection:
Given the image below (after applying the transformation), those are the stages we do:



1. We smooth the image with a convolution

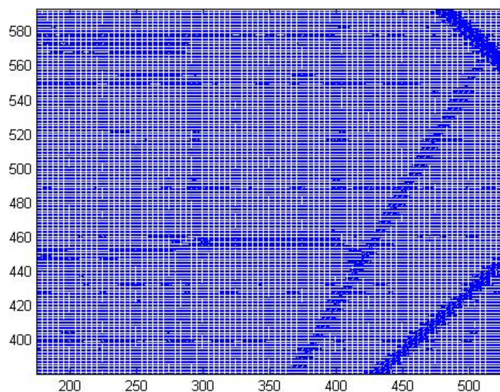


   function
2. We calculate the gradient of the smoothed image, and hide the camera and the two bars that hold the camera

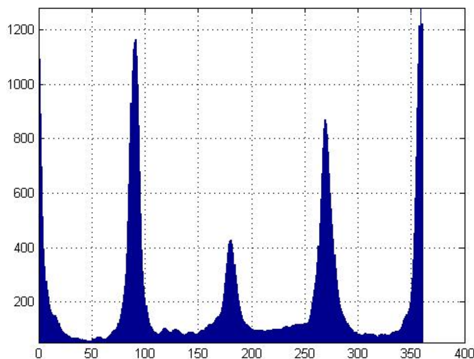


this is the quiver of the gradient (in x,y direction):

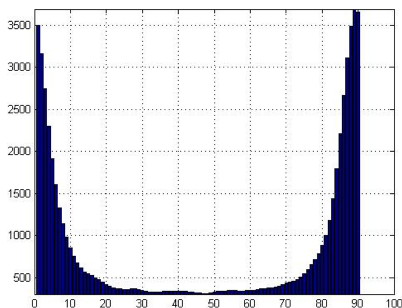
Link to this doc: http://goo.gl/kVkbV
STC Algorithm paper: http://goo.gl/hrrTA (http://www.springerlink.com/content/n5770672r3852113/fulltext.pdf)



3. We calculate the angle (atan2) between the gradients in x and y direction (smoothing it to circularity)



4. We “reshape” the histogram, to show only angles from 0-90 degree (we combine the angles together, (0-90, 91-180, 181,270, 271,360)



5. We take the maximum value of the histogram, which is the right angle (In the picture 90 degree)

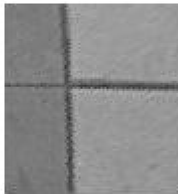
Rotation and Calibration

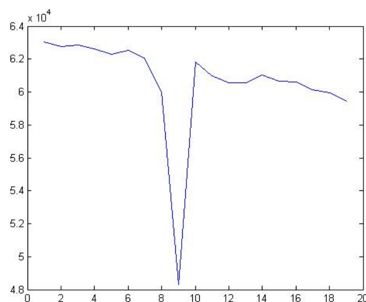
For the robot to rotate 90 degrees, we send it a rotate command, specifying the time, T, we want it to spend in rotation. We initially set T to a time that let’s the robot do the first rotation accurate enough.
After each rotation (including the first), we take an image of the area, and using the angle

detection algorithm described above, we detect the robot's deviation. If the robot has done the rotation accurate enough, no calibration is needed. Otherwise, let d be the angle that the robot actually rotated in (calculated using the angle detection algorithm). Now we know that in the next time, we should give the robot 90/d * T time to do the rotation, because in the time slice T, it rotated d degrees, so for 90, it needs a time slice of 90/d * T. So we reassign T to get the new value and send the robot another command in order to let it complete the 90 degrees rotation it was supposed to perform.

Obstacles Detection

We defined obstacles as completely white cells, whereas the board cells are gray.
Thus, summing up the values of all the pixels in an obstacle cell should give us a higher value than the sum of all the values of the pixels of a regular free cell.
This way we can know whether a cell is a free or an obstacle cell.
In order to detect the obstacles in the best possible way, we had to know where is our location exactly, in order to do this, we summarized the lines of a specific region of the picture (as shown below), (after applying the transformation), and we take the minimum value, this, will be the line between the two cells



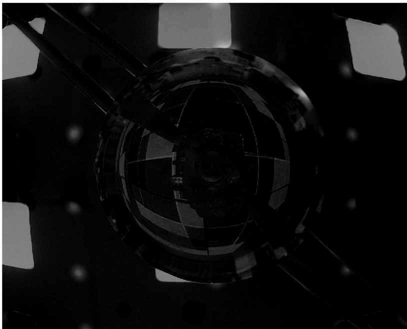


(this is the horizontal line in the above picture)

Connectivity Issues

One of the obstacles we faced during our work with the robot is connectivity issues.
Since the connection to the robot and the camera is wireless and because there are other wireless broadcasters in the area in the same frequency, in some cases, commands sent to the robot from the computer had not reach the robot. In other cases, the command is received by

the Robot, but the Robot's response didn't reach the computer, or a check-sum error occurred. Therefore, we send the command repetitively until we get a reply from the robot that the command was received successfully.
Initially, rotating and driving forward were done via timing. We send the robot a rotate/drive command, wait the needed time to perform the operation and then send a stop command. Therefore, because of the connectivity issues, the robot might rotate too much, or drive too much. This happened frequently and in order to avoid this, atomic time commands were added to the robot board. For example, with the rotate clockwise command, we sent the robot the time it should spend in the rotation.
However, this creates another problem, when the robot receives our first command, but his reply does not reach out to the computer, the computer will send the command again. Thus, instead of 90 degrees rotation, 180 degrees rotation might occur.
To solve this issue, we implemented a function that receives two photos, compares them and tells whether or not they were shot from the same position. This was implemented by subtracting the pixel matrix of the second photo from the first photo, and then summing up all the pixels. If we got a small sum, this means that the photos are almost identical, i.e. were shot from the same position - no rotation between them.
So, we take a photo before sending a command. Before re-sending a command again (in case the robot did not reply with success), we take another photo. Using the function described above, we can compare the two photos and see if the robot performed an action in response to the sent command. If yes, we do not re-send the command.



example of comparing two pictures, with 90 degree rotation between them, clearly we can see that the robot did the rotation successfully.

Videos

http://www.youtube.com/watch?v=x-9XfCX2xfs