Self-Mapping Mobile Robot (MapBot)

Senior Capstone Project Report

Submitted to

Dr. Aleksander Malinowski and Dr. Winfred Anakwa

by Stephanie Luft

9 May 2006

Abstract

This report discusses the results of a feasibility study of a Self-Mapping Mobile Robot. The objective of the project is to develop a robot that will map its environment and locate itself on the map. Topics covered will include descriptions of the laser distance-measuring system, including relevant calculations and image processing algorithms, and the probabilistic mapping system, including the robot and its associated software and hardware. The results of the feasibility study, as well as future areas of opportunity will be also be presented.

Table of Contents

Introduction

Objective

The objective of the Self-Mapping Mobile Robot project is to develop a robot that can independently create a map of its environment, locate itself on the map, and orient itself within the environment.

Applications

A robot of this type has various uses. This technology could be useful on military robots such as the PackBot, which need to dropped into unknown terrains with the ability to map and navigate. A similar technology is already in use on household robots like the Roomba and the Scooba, and could be adapted for use on a Mars or Moon rover.

System Description

Block Diagram and Functional Description

The robot system, shown in Figure 1 below, will consist of the Pioneer 2 mobile platform, previously used in the GuideBot project, a laser distance meter, and a laptop computer currently mounted on the mobile platform. The system may include a remote computer in addition to the laptop for running calculations, storing map data, and interfacing with a user.

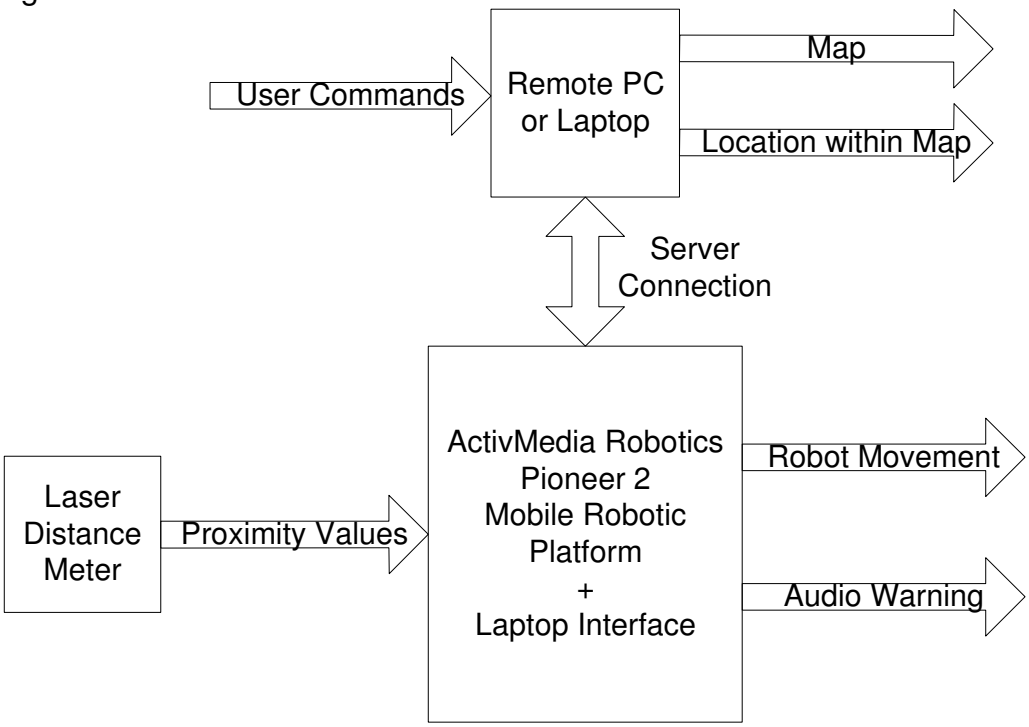


Figure 1: System block diagram for the MapBot

The robot will move throughout its environment at the command of a user or the software. It will use data from the laser distance meter to create a map of its environment and locate itself on the map. The robot will be able to make an audio warning sound if it encounters a problem such as low battery power.

The core of the robot is the ActivMedia Pioneer 2 robotic platform. It has three wheels, a set of ultrasonic sensors, and the batteries that power the robot and all of the equipment that it carries. The user enters commands through the laptop or a remote computer, which connects to the robot through a TCP/IP server.

The laser distance meter became a much larger portion of the project than originally anticipated. This is a result of the unavailability of meters that met the needs of the project in terms of both functionality and budget. The kind of laser meters that do meet the requirements cost thousands of dollars and would add up to one-half kilogram to the weight of the robot. There are much less expensive "point and shoot" models available at hardware stores, however they cannot be controlled through the computer. Therefore, a different type of laser meter had to be created for the project. The one in use at this time is constructed of a standard laser pointer from Radio Shack, and a USB2 webcam, available at most electronics stores. It is mounted on a Pan-Tilt Unit, which allows it to turn in a 320-degree arc, as well as tilt up and down.

The outputs of the system are the map and the robot's location, which are displayed on the laptop screen. The robot can drive forward and in reverse, and rotate right or left at the user's command. Finally, the robot also emits an audio warning to signal problems such as a low battery or an obstacle blocking its path.

Functional Description

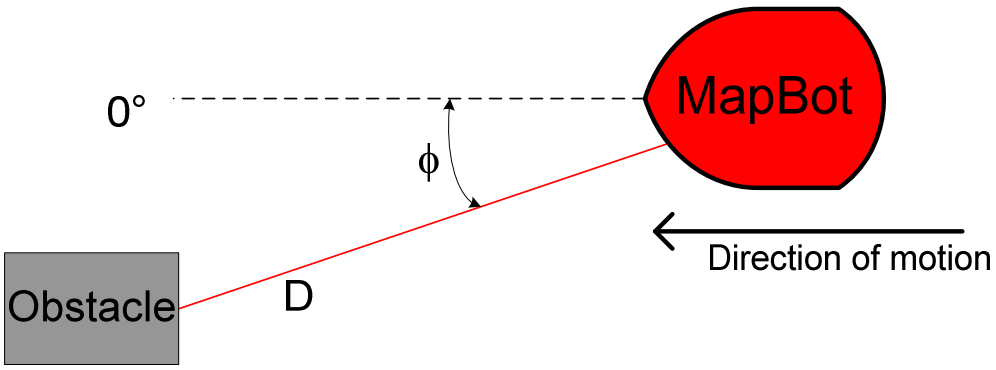


Figure 2: Diagram of measuring and mapping principle

The laser distance meter will rotate through nearly 360 degrees, recording distance measurements and the angle of rotation, ϕ. After the rotation has been completed, the map will be created and displayed on the laptop. At that point, the robot can decide to move to a sparse area of the map to fill it in with additional measurements, or it can wait for a user command.

Software

Functional Modes

The robot has two main functional modes, Mapping and Maneuvering. In the Mapping mode, the robot operates independently to map its environment and locate itself on the map. This mode contains several functionalities:

- Distance reading
- Obstacle avoidance
- Self-locating – determining the robot's location on the map
- Plotting the visual representation of the robot's environment and showing its own location
- Interfacing – allowing the user to command the robot to map an area and displaying the map to the user.

In the Maneuvering mode, the robot is controlled by the user. It is entirely the user's responsibility in this mode to protect the robot from collisions with obstacles or other dangers. This mode's functionality is similar to that of the Mapping mode:

- Interfacing – allowing the user to command the robot to move or adjust the Pan-Tilt Unit.
- Distance reading
- Self-locating

The functionality described above is implemented in MATLAB, Simulink, and C++. The TCP/IP servers that connect to the robot and to the Pan-Tilt Unit to control the movement of each one were written in C++ by Dr. Malinowski. Simulink provides the interface to the webcam in a very simple "plug and play" kind of format. This is the factor that drove the decision to use MATLAB for the rest of the programming in the project. MATLAB is very easy to use for image processing and creation of graphics or plots, however it can be significantly slower than C++ in certain applications. That said, the current software is only slowed by the interface to the server and, most importantly, the interface to the webcam. The mapping and other processing are relatively efficient.

Flowcharts

The flowcharts in Figure 3 show the processes for the main routines in the MapBot project. The actual code can be found in the Appendix of the report.

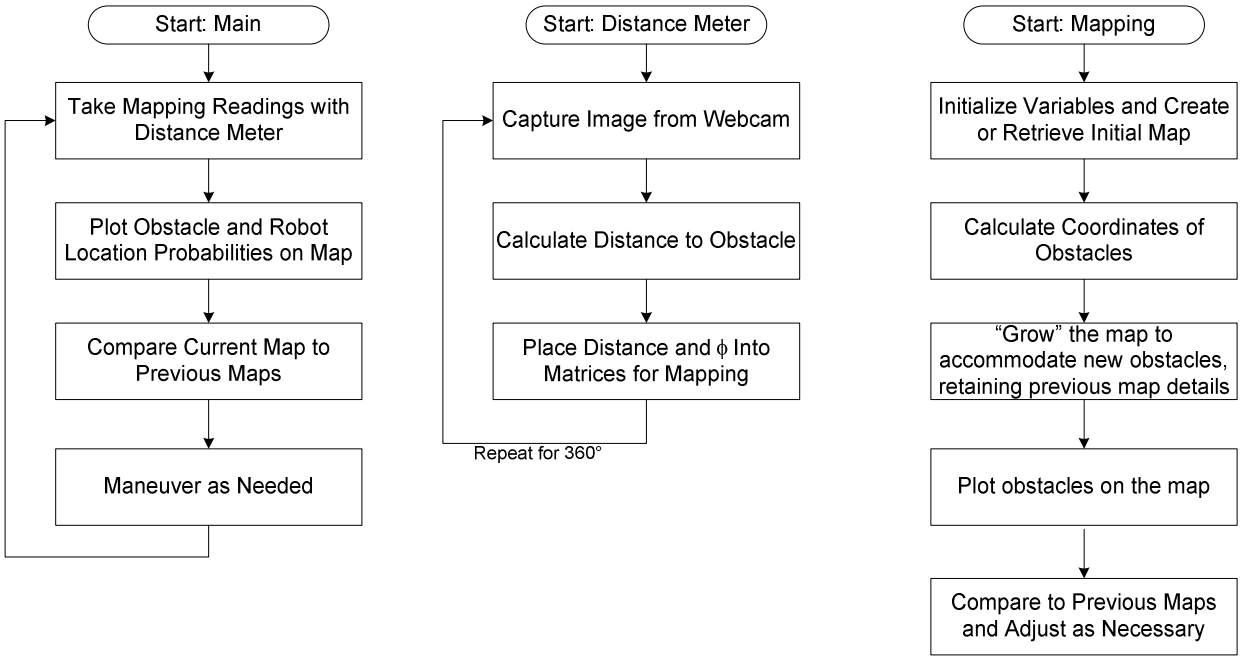


Figure 3: Software flowcharts for MapBot

Design Process

Theory

Laser Distance Meter

The concept behind the laser distance meter was one previously used at Drexel University. The laser pointer shines on a target a certain distance *D* away from the robot, as shown in Figure 4 below. This target is the obstacle, such as a wall, that the robot will map. The camera is separated from the laser pointer by a distance *h* and takes a snapshot of the scene in front of it, including the laser dot on the obstacle. The image processing software then detects the brightest pixel in the frame, which is the center of the laser dot.

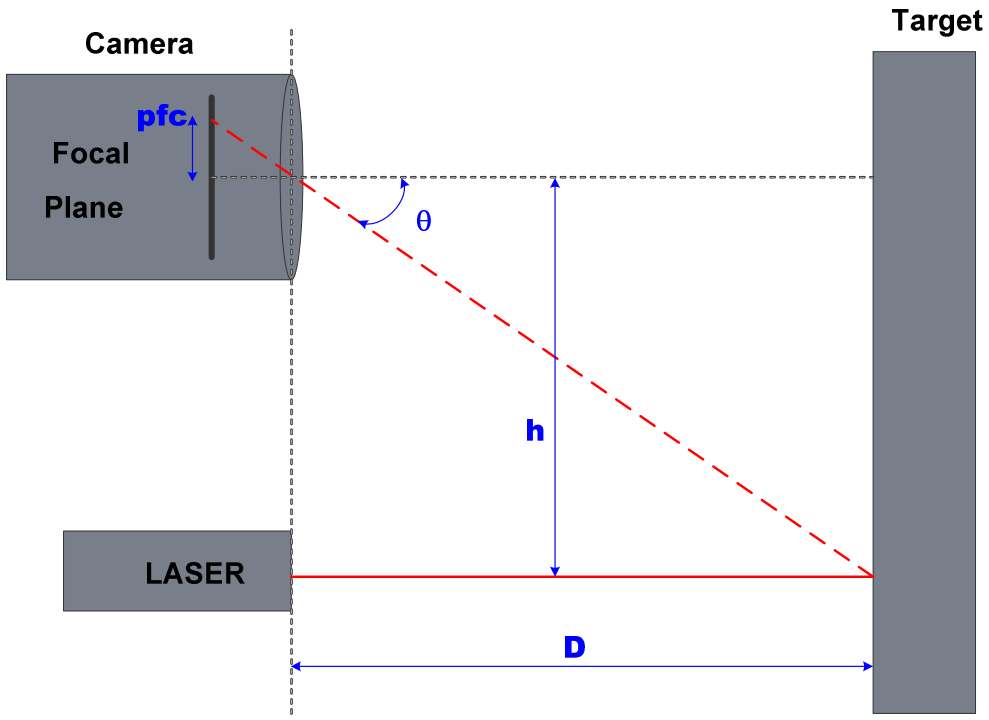


Figure 4: Laser Distance Meter Diagram

The lines that result from this setup create two similar triangles; the base of the second one is the number of pixels from the center of the focal plane, marked *pfc* in the diagram. Using right triangle trigonometry, it is possible find the angle θ from the value of *pfc*, and from there find the distance *D*, outlined in the equations below.

Guiding Equation: $D = \frac{h}{\tan(q)} = \frac{h}{\tan(pfc*m+b)}$.

D = distance (meters) to object
h = distance (meters) between laser beam and center of lens
pfc = pixels from center of image to laser dot
m = calibration coefficient
b = calibration offset

The laser distance meter was calibrated by taking set measurements and determining the *pfc* from the image processing routine. The value of *h* was pre-set to allow for the optimal measuring range to occur between 1 and 20 meters. The values of *pfc* and *D* were then substituted into the equation above, and *m* and *b* calculated. With an initial set of values determined, the user is able to use an iterative process to "tweak" the calibration coefficient and offset until the desired level of accuracy in the desired range is achieved. In this case, the calibration variables were set to yield the lowest average error across the range of interest. However, they could have been set to give even greater accuracy over a smaller range, if desired.

Results of Calibration Data show that the values of the equation are

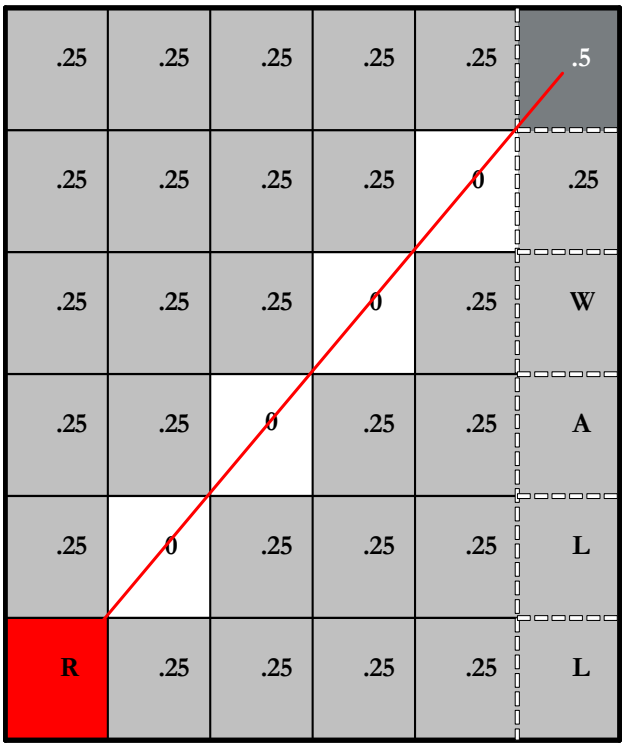
h = 0.189
m = 0.00113923
b = -0.0324705

which leads to a final equation of

$$D = \frac{0.189}{\tan(pfc*0.00113923-0.0324705)}.$$

Mapping Algorithm

Once all of the distance readings have been taken, the obstacles are plotted on the map. A probabilistic algorithm was used, so that as opposed to plotting simple obstacles, the program instead plots the probability that there is an obstacle in a particular location. This allows the map to adapt to changes in the environment, such as a person walking through a room or a chair moved from one location to another in a hallway.



Pixel values on the map range from 0 (white) to 1 (black). All pixels begin at a light gray color, which has a probability of about 1 in 4. When an obstacle is detected, the pixel's probability value is incremented by 0.25. When an area is determined to be clear space, i.e. the pixels that the laser beam passes through unobstructed, then the probability values of the pixels in the area are decremented by 0.25. In this way, temporary obstacles are phased out of the map while permanent obstacles are made clearer.

Figure 5: Demonstration of Probabilistic Mapping Scheme

Simulation

The initial mapping algorithm was tested with simulated data. The data used reflected a perfectly rectangular room, with no additional obstacles and the robot sitting at one end of it. In this way, it was demonstrated that the mapping algorithm worked for a simple structure.

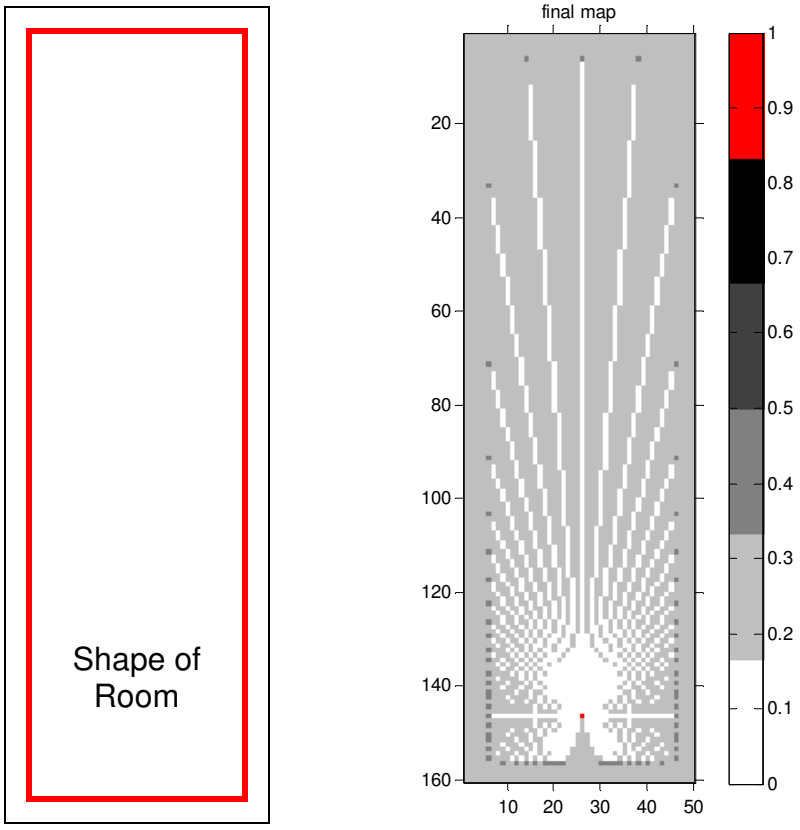


Figure 6: Simulated room and the map created with simulated data

Testing

MapBot was tested at the Student Scholarship Expo, where it detected obstacles such as a chair, a table, two walls, and two students standing nearby, as shown in Figure 7 on the following page. The results show that the mapping algorithm did a good job of picking out the obstacles; however there were two important issues. First, the map had a left-to-right mirror effect. Second, there are several points lying very far outside of the mapped area, probably as a result of something being too close to the distance meter. If an object is too close to the robot, the snapshot taken by the webcam does not include the laser dot, and so readings may be wildly inaccurate.

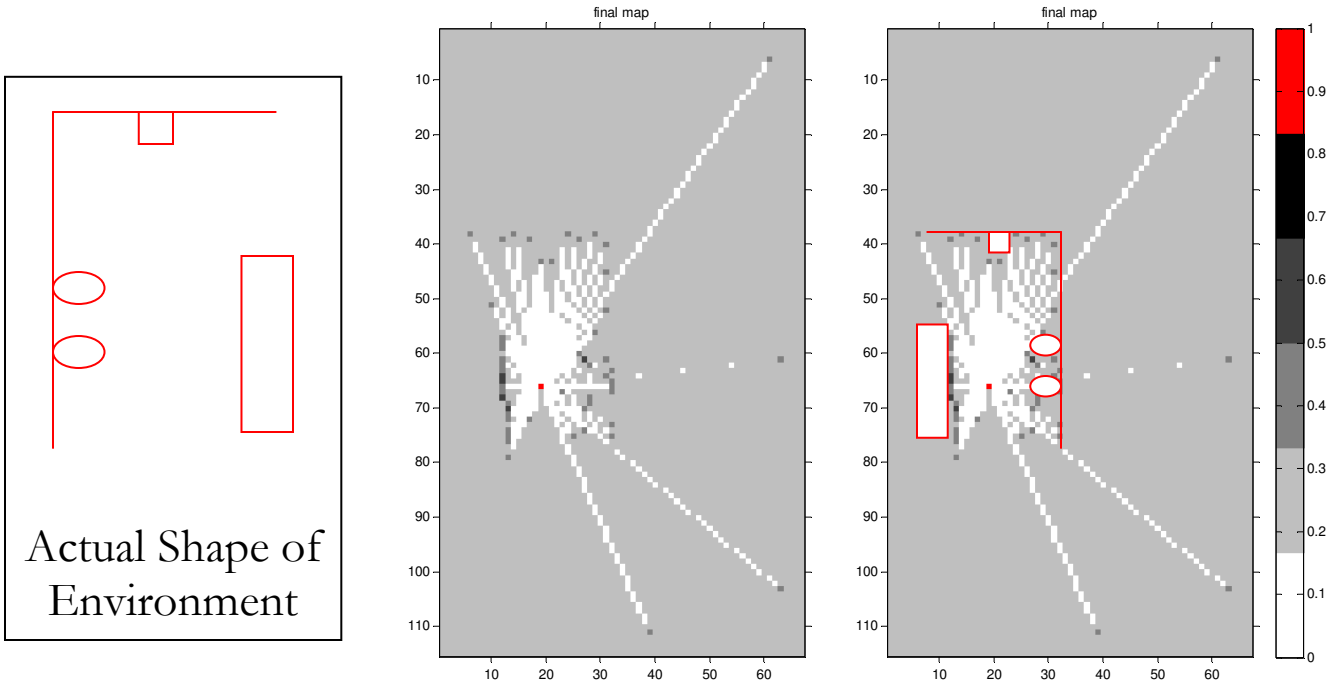


Figure 7: Testing the mapping process. Two issues: outliers and horizontal mirror effect.

After the Expo, the mirror image effect in the plotting routine was corrected, and statistical filtering was implemented in the data calculating routine to prevent this type of outlying measurement.

Results

The results of the laser distance meter testing were good, especially in the middle distances from about 2 to 8 meters, where there was generally less than a 2% error between calculated and measured data. The highest error was at the ends, up to 12%. The absolute maximum range of the sensor is ½ to 175 meters, however to measure with any accuracy, the range is capped closer to 20 meters.

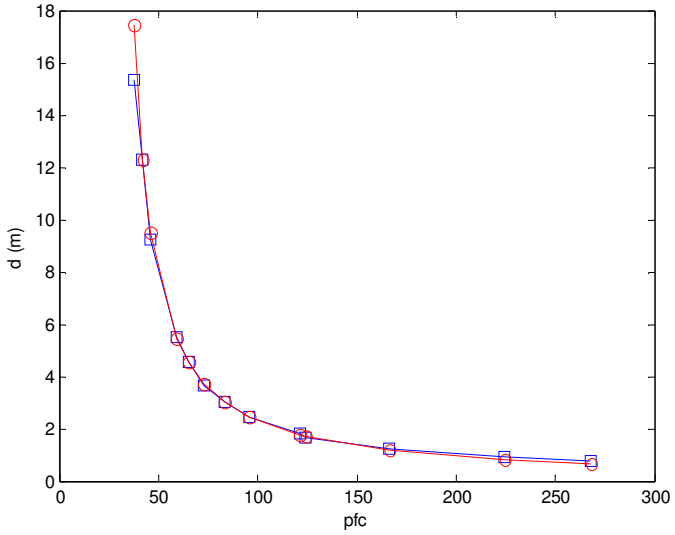


Figure 8: Plot of Actual (circles) vs. Calculated (squares) Distance Data

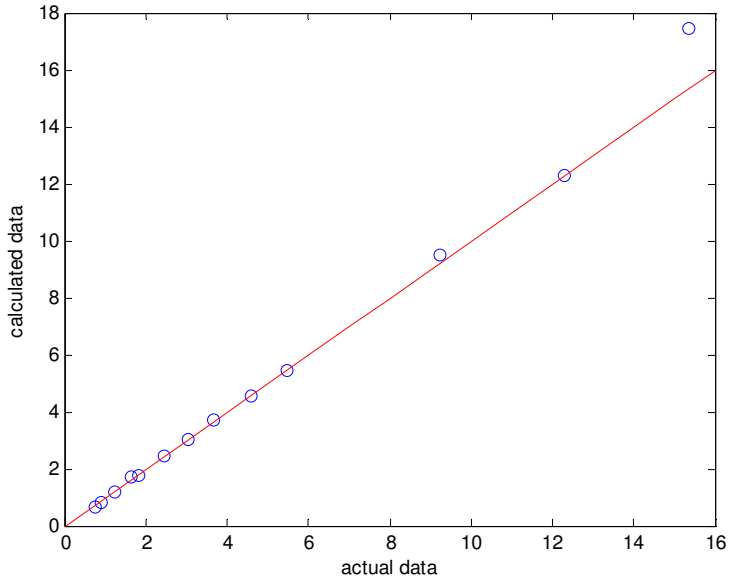


Figure 9: Plot of Actual vs. Calculated Distance Data, showing nearly a +1 correlation coefficient.

Conclusion

At the end of the project, MapBot has met the original objectives of mapping its environment and locating itself on the map. The mapping algorithm is complete to the point of allowing the robot to take multiple mappings from the same location. The program and the user both have complete control over the pan-tilt unit that turns the laser meter. The robot is also capable of responding to user commands to move and turn, both directly from the laptop as well as remotely through a workstation PC.

Future Work

Other teams may be interested in pursuing advances to this project in areas such as

- Web control for remote users
- Navigation and advanced decision-making in regard to paths of motion, possibly including semi-random motion for maneuvering around obstacles
- Greater distance meter accuracy and enhanced ability to measure in bright light
- The ability to map in three dimensions and to detect stairs
- Efficiency and conservation of battery power.

References

- GuideBot Capstone Project 2005 by John Hathway and Daniel Leach. http://cegt201.bradley.edu/projects/proj2005/guidebot

- Laser Meter: from Drexel University. http://www.pages/drexel.edu/~twd25/webcam_laser_ranger.html

- Mapping: from Dartmouth University, 1999. http://www.jonh.net/%7Ejonh/robots/mapping/submitted-paper.html

- Thesis: Duckett, Thomas. "Concurrent Map Building and Self-Localization for Mobile Robot Navigation". Manchester, United Kingdom.

Appendix: MATLAB Files

Createmap.m

```
% Create a Map
% 12 April 2006     MapBot Project    Stephanie Luft
% This program is the main module for the mapping algorithm.
% It could be turned into a function, as shown.  Right now, to create the
% first map, the 'maparray=finalmap;' line must be commented out, and to
% create subsequent maps, the 'firstmap' line must be commented out.


%function [finalmap]=createmap(maparray)

runPTU
robotcolor
firstmap
%maparray=finalmap;
[ploti,plotj]=obstaclecoords(phi, distance, roboti, robotj);
figure,plot(ploti,plotj,'bo',roboti,robotj,'ro')    % plot the data points on
a graph
[newmap,ii,jj]=growmap(ploti,plotj,roboti,robotj,maparray);
[finalmap,roboti,robotj]=plotobstacle(ploti,plotj,newmap,roboti,robotj,ii,jj)
;
figure,imshow(finalmap),title('final map'),colormap(botcolor),axis
on,colorbar
    % Plot the new map
```

runPTU.m

```
% Control the PTU and take distance readings
% 11 April 2006     MapBot Project    Stephanie Luft
% This program communicates with the PTU server to control its movements
% and take distance readings every 5 degrees.  It requires that the
% tcp_udp_ip directory to be placed in the same folder as the directory
% holding runPTU.m


% Add PTU path
addpath('..\tcp_udp_ip')

% Open PTU connection
connection=pnet('tcpconnect','localhost',8082);
if connection==-1
    disp('ERROR: CONNECTION TO PTU NOT MADE')
    return      %break program after error
end
str=pnet(connection,'readline');

% Calibrate PTU
pnet(connection,'printf','PP0\n')
str=pnet(connection,'readline');
```

```
if str(1)~= '+'
    disp('ERROR: INVALID COMMAND OR OTHER PROBLEM WITH PTU')
    disp(str)
end
pnet(connection,'printf','TP0\n')
str=pnet(connection,'readline');
if str(1)~= '+'
    disp('ERROR: INVALID COMMAND OR OTHER PROBLEM WITH PTU')
    disp(str)
end

% Move PTU Number of Positions to Phi (-160 to +160 degrees)
phistep=-160:5:160;
[m,n]=size(phistep);
for i=1:n
    position=round(phistep(i)*19.4445);     % step 5 degrees
    if position>3090
        position=3090;
    elseif position<-3089
        position=-3089;
    end
    pos=int2str(position);
    string=strcat('PP',pos,'\n');
    pnet(connection,'printf',string)
    str=pnet(connection,'readline');
    if str(1)~= '+'
        disp('ERROR: INVALID COMMAND OR OTHER PROBLEM WITH PTU')
        disp(str)
    end

% Run ImageCapture
   sim('ImageCapture')
    pic=red(:,:,2)+green(:,:,2)+blue(:,:,2);
        % Have to use 2nd set of data because 1st tends to come out black

% Calculate Distance to Obstacle
    dist=finddistance(pic);

% Record Distance and Phi
    distance(i)=dist;
    if phistep(i)<=0
        phistep(i)=360-abs(phistep(i));
    end
    phi(i)=phistep(i);
end

% Reset PTU
pnet(connection,'printf','PP0\n')
str=pnet(connection,'readline');
if str(1)~= '+'
    disp('ERROR: INVALID COMMAND OR OTHER PROBLEM WITH PTU')
    disp(str)
end
pnet(connection,'printf','TP0\n')
str=pnet(connection,'readline');
if str(1)~= '+'
```

```
    disp('ERROR: INVALID COMMAND OR OTHER PROBLEM WITH PTU')
    disp(str)
end

% Close connection
pnet(connection,'close')
```

finddistance.m

```
% Find distance
% MapBot Project    Stephanie Luft
% This function finds the distance to an obstacle from the picture taken by
% the webcam.  If the pixels from the center (pfc) is outside of the range
% permitted by the equation, the distance is set to -1. Later routines flag
% this as an anomoly before plotting.

function [distance]=finddistance(pic)

pfcpic=pic(250:258,1:320);  % Look at the window in the picture where the
laser dot is expected to be
maxval=0;
pfc=0;
for i=1:6
    for j=1:320
        if pfcpic(i,j)>maxval   % if the new pixel is the brightest so far
            pfc=320-j;          % then calculate pfc
            maxval=pfcpic(i,j);
        end
    end
end
distance=.189/tan(.0013923*pfc-.0324705);
    % this equation was found as the result of testing with the laser
distance
    % meter.  If a different setup were used, or for higher accuracy in
different
    % ranges, a different equation or different coefficients would be used.
if pfc<31 | pfc>282     % if pfc is outside of plausible range
    distance=-1;
end
```

robotcolor.m

```
% Set up the colormap for the maps
% 30 March 2006     MapBot Project    Stephanie Luft
% This program sets up the colormap for all of the maps.  It is redundant
% to the firstmap program, however firstmap.m is not used for second
% mappings, so I've left it in.

obstacle = 9/12;
maybe3 = 7/12;
maybe2 = 5/12;
maybe1 = 3/12;
space = 1/12;
robot = 11/12;
botcolor = [1 1 1; .75 .75 .75; .5 .5 .5; .25 .25 .25; 0 0 0; 1 0 0];
```

firstmap.m

```
% Set up the first map
% 30 March 2006     MapBot Project    Stephanie Luft
% This program sets up the first map, a very basic 20x20 pixel array with
% the robot in the center.  This provides a basis for expansion for future
% mappings.  It uses the robotcolor colormap.

obstacle = 9/12;
maybe3 = 7/12;
maybe2 = 5/12;
maybe1 = 3/12;
space = 1/12;
robot = 11/12;
botcolor = [1 1 1; .75 .75 .75; .5 .5 .5; .25 .25 .25; 0 0 0; 1 0 0];
maparray=ones(20,20)*maybe1;
roboti=10;
robotj=10;
maparray(roboti,robotj)=robot;
```

obstaclecoords.m

```
% Find the coordinates of the obstacles
% 30 March 2006     MapBot Project    Stephanie Luft
% This program finds the coordinates of the obstacles that will be plotted
% on the map, from the distance and angle data.

function [ploti,plotj]=obstaclecoords(phi, distance, roboti, robotj)

scale=10;   % scale: 1m = 10 pixels
ploti=0;    % if these remain zero, it will cause an error later
plotj=0;
i=0;
index=0;
dist=[0];
[m,n]=size(phi);
```

```
for j=1:n
    if distance(j)>0 & distance(j)<20   % check for wild points
        index=index+1;
        dist(index)=distance(j);
        p(index)=phi(j);
    end
end
meand=mean(dist);
stdd=std(dist);
for j=1:index
    if dist(j)<meand+stdd & dist(j)>meand-stdd  % another check for wild
points
        quadrant = floor(p(j)/90);
        i=i+1;
        if p(j)==0 | p(j)==360
            ploti(i)=round(roboti-dist(j)*scale);
            plotj(i)=robotj;
        elseif p(j)==90
            ploti(i)=roboti;
            plotj(i)=round(robotj-dist(j)*scale);
        elseif p(j)==180
            ploti(i)=round(roboti+dist(j)*scale);
            plotj(i)=robotj;
        elseif p(j)==270
            ploti(i)=roboti;
            plotj(i)=round(robotj+dist(j)*scale);
        elseif quadrant==0
            ploti(i)=roboti-round(dist(j)*cos(p(j)*pi/180)*scale);
            plotj(i)=robotj-round(dist(j)*sin(p(j)*pi/180)*scale);
        elseif quadrant==1
            p(i)=p(i)-90;
            ploti(i)=roboti-round(dist(j)*cos(p(j)*pi/180)*scale);
            plotj(i)=robotj-round(dist(j)*sin(p(j)*pi/180)*scale);
        elseif quadrant==2
            p(i)=p(i)-180;
            ploti(i)=roboti-round(dist(j)*cos(p(j)*pi/180)*scale);
            plotj(i)=robotj-round(dist(j)*sin(p(j)*pi/180)*scale);
        elseif quadrant==3
            p(i)=p(i)-270;
            ploti(i)=roboti-round(dist(j)*cos(p(j)*pi/180)*scale);
            plotj(i)=robotj-round(dist(j)*sin(p(j)*pi/180)*scale);
        end
    end
end
```

growmap.m

```
% Expand the previous map to accomodate new obstacles
% 5 April 2006     MapBot Project    Stephanie Luft
% This program grows the previous (or initial) map to allow space for the
% obstacles to be plotted.  It provides for a 5-pixel border around each
% edge.

function [newmap,roboti,robotj]=growmap(ploti,plotj,roboti,robotj,maparray)

maparray(roboti,robotj)=0;
imax=max(ploti);
jmax=max(plotj);
imin=min(ploti);
jmin=min(plotj);

imax2=imax-roboti;
jmax2=jmax-robotj;

if imax<0 imax=0; end
if imin>0 imin=0; end
if jmax<0 jmax=0; end
if jmin>0 jmin=0; end

[m,n]=size(maparray);

% grow the map
if imin<0   % expand in i direction above robot
    for i=1:m
        for j=1:n
            newmap1(i+abs(imin)+5,j)=maparray(i,j);
         end
    end
    roboti=roboti+abs(imin)+5;
else
    newmap1=maparray;
end
if jmin<0   % expand in j direction behind robot
    [m,n]=size(newmap1);
    for i=1:m
        for j=1:n
            newmap2(i,j+abs(jmin)+5)=newmap1(i,j);
        end
    end
    robotj=robotj+abs(jmin)+5;
else
    newmap2=newmap1;
end
[m,n]=size(newmap2)

newmap3=newmap2;    % expand in the two remaining "positive" directions
a=imax2+roboti+5;
b=jmax2+robotj+5;
newmap3(a,b)=0;
```

```
% fill in the gap
newmap=newmap3;
newmap(1:abs(imin)+5,:)=3/12;
newmap(:,1:abs(jmin)+5)=3/12;
newmap(m:a,:)=3/12;
newmap(:,n:b)=3/12;

% figure,imshow(newmap1),title('newmap1'),axis on
% figure,imshow(newmap2),title('newmap2'),axis on
% figure,imshow(newmap3),title('newmap3'),axis on
% figure,imshow(newmap),title('newmap'),axis on
```

plotobstacle.m

```
% Plot the obstacles on the map
% 6 April 2006     MapBot Project    Stephanie Luft
% This program plots the obstacles from the coordinates calculated in
% obstaclecoords.m on the expanded map created by growmap.m

function
[finalmap,roboti,robotj]=plotobstacle(ploti,plotj,newmap,roboti,robotj,ii,jj)

[p,n]=size(ploti);
[x,y]=size(newmap);

if ii~=roboti | jj~=robotj  % ii and jj are recalculated values for roboti
and robotj
    roboti=roboti+abs(min(ploti))+5;
    robotj=robotj+abs(min(plotj))+5;
    ploti=ploti+abs(min(ploti))+5;
    plotj=plotj+abs(min(plotj))+5;
end

% plot obstacles and clear spaces
for i=1:n
    if newmap(ploti(i),plotj(i))<4/6
        newmap(ploti(i),plotj(i))=newmap(ploti(i),plotj(i))+1/6;
    end
    if roboti==ploti(i)    % slope=infinity
        if robotj>plotj(i)
            newmap(roboti,plotj(i)+1:robotj-
1)=newmap(roboti,plotj(i)+1:robotj-1)-1/6;
        elseif robotj<plotj(i)
            newmap(roboti,robotj+1:plotj(i)-
1)=newmap(roboti,robotj+1:plotj(i)-1)-1/6;
        end
    else
        m=(robotj-plotj(i))/(roboti-ploti(i));
        c=(roboti*plotj(i)-robotj*ploti(i))/(roboti-ploti(i));
        if ploti(i)>roboti & plotj(i)>robotj
            for a=1:x
                for b=1:y
                    if a>roboti & a<ploti(i) & b>robotj & b<plotj(i)
```

```
                if b==round(a*m+c)
                    if newmap(a,b)>1/6
                        newmap(a,b)=newmap(a,b)-1/6;
                    end
                end
            end
        end
    end
elseif ploti(i)>roboti & plotj(i)<robotj
    for a=1:x
        for b=1:y
            if a>roboti & a<ploti(i) & b<robotj & b>plotj(i)
                if b==round(a*m+c)
                    if newmap(a,b)>1/6
                        newmap(a,b)=newmap(a,b)-1/6;
                    end
                end
            end
        end
    end
elseif ploti(i)<roboti & plotj(i)>robotj
    for a=1:x
        for b=1:y
            if a<roboti & a>ploti(i) & b>robotj & b<plotj(i)
                if b==round(a*m+c)
                    if newmap(a,b)>1/6
                        newmap(a,b)=newmap(a,b)-1/6;
                    end
                end
            end
        end
    end
elseif ploti(i)<roboti & plotj(i)<robotj
    for a=1:x
        for b=1:y
            if a<roboti & a>ploti(i) & b<robotj & b>plotj(i)
                if b==round(a*m+c)
                    if newmap(a,b)>1/6
                        newmap(a,b)=newmap(a,b)-1/6;
                    end
                end
            end
        end
    end
else
    for a=1:x
        for b=1:x
            if a>roboti & a<ploti(i)
                if b==c
                    if newmap(a,b)>1/6;
                        newmap(a,b)=newmap(a,b)-1/6;
                    end
                end
            elseif a<roboti & a>ploti(i)
                if b==c
                    if newmap(a,b)>1/6;
                        newmap(a,b)=newmap(a,b)-1/6;
```

```
                        end
                    end
                end
            end
        end
    end
end
newmap(roboti,robotj)=1;
finalmap=newmap;
```

ImageCapture.mdl

*Note: This requires the Simulink Image Acquisition Toolbox