Autonomous Navigation and Mining

System Manual

Emily Biller

James Ferguson

Scott Hamilton

John Lucas

Travis Williams

Instructor: Dr.Yenumula Reddy

Faculty Sponsor: Dr. Powsiri Klinkhachorn

Abstract

The Robotic Autonomous Operation (RAO) system includes sensory hardware and control software and will be where our research will benefit. This allows the robot to localize itself inside of the allocated area and complete mining operations. In last year's competition, the robot was extremely behind schedule in reference to the mechanical system structure. This caused delays in the development of the RAO system to the point where it could not even be tested before competition.

Our group's direct involvement with the robotics team is to develop an autonomously mining algorithm in Matlab to test on the iRobot Create. This system is to be developed with the full cooperation of the West Virginia University Robotics Team. The previous work of the team will be reviewed, but development will start from scratch as the past RAO system was never tested nor proved.

Table of Contents

Introduction

This project originated due to the NASA Mining competition, formally known as Lunabotics. West Virginia University has participated in the NASA Mining Competition for the past three years ever since it was recommended they compete by alumni astronaut, Jon McBride. During the past three years, the university has done exceedingly well but has not yet won a first place trophy. The competition is heavily weighted toward autonomous operations, but no team has been awarded the full amount of points for full autonomy.

Autonomous operation can be described best by using a series of states. These states include but are not limited to orientation, traversing, mining, and dumping. The listed states are ones which will not change as a robot changes in size or design allowing this software and architecture to be developed while possible changes occur. This is a major factor in case parts fail or, for example, this autonomous operation is to be implemented on earth for construction companies.

Design Achievements

We accomplished full autonomy of the mining operation with the iRobot, which was different than what we originally planned in the fall semester. In the fall, we planned on working with the actual mining robot. Since the robot was being configured for parts and changes were being made to the design, it wasn't possible for us to test with it. This brought about the use of the iRobot and instead of coding in C# we coded with Matlab to test. With those changes to our plans we developed a fully autonomous mining operation simulation using the iRobot.

Complete Hardware Design

List of Materials

Hokuyo Range Finder

The Hokuyo Laser Range Finder is a semiconductor laser diode. The Sensor polls 683 different data points and returns the distance away from objects along those points. The range finder is accurate from a distance of 20mm-4000mm and ha a 100ms scanning time. The Range

finder scans along a 240 degree arc of detection. This sensor is used for localization. Using the range finder the robot can determine where it is in the arena and then use that data to determine where it needs to go and what it needs to do. The points that are polled from the arc can be set and manipulated in the software and more points can be added to the 7 points that are being used in the algorithm.

iRobot Create Programmable Robot

The iRobot Create Programmable Robot is a robot that is very easy to configure and program using MATLAB software. The robot features two drive wheels giving full movement and allowing for precision movements. The robot also features onboard sensors that are not used in the implementation of the algorithm. The sensors included are a Lidar and a Touch sensor. The Hokuyo will be taking the place of the on board Lidar for the purposes of simulating the algorithm used on the competition robot. The robot allows for custom written software and is used for quick debugging.

Complete Software Design

Currently the autonomy system is produced for the 2014 NASA Robotic Mining Competition and not as a commercial product. The software is a finite state machine with four states running on a continuous cycle. The four states are Driving, Mining, Return, and Deposit. The software begins the competition by locating itself in the arena and then finding the center of the arena. In order to accomplish this, the robot polls the locations using the sensor and determines where it needs to go to be in the center of the arena to begin the traversal of the arena. Once the robot has gotten relatively close to the center of the arena it then begins the drive state where it fully centers itself. In order to maintain a path along the center of the arena the algorithm polls two locations along the wall and creates a right triangle using these sensors. With these data points the robot tries to keep itself in the ideal position to create a right triangle of desired proportions.

Once the robot has successfully navigated through the center of the arena to the mining area the state then changes to mining. The mining operation begins by rotating clockwise to find a 45 degree angle and then mining at that angle off of the center. The robot takes small trips mining and incrementing the counter until it reaches the maximum limit as se in the code. Currently, the limit is three. After mining three times in the first location the robot then begins the traversal back to the starting side of the arena to find the deposit bin. The robot uses the same algorithm to travel backwards across the arena as it does to traverse it the first time, but drives in reverse the entire way to save time. Once the robot is close enough to the boundary it begins the dumping operation.

The dumping operation is very small movements forward and backward to simulate trying to dump all material out of the bin to get as much deposited as possible. Once the bin is fully emptied the robot them changes state to drive across the arena attempting to mine again. Once the robot reaches the mining area it mines directly in front of it instead of the previously used angle. The same mining operation is carried out until the robot is full. The robot then returns to the starting area of the arena to deposit the material that has been mined. The deposit state is then entered and the material is deposited into the bin. Once empty the robot then traverses the arena for the third time forward until it reaches the mining area.

Once in the mining area the robot then turns counter clockwise until it reaches an angle similar to the first mining operation in order to mine in a 3rd location. The same mining procedure occurs and then the robot enters the return state. The robot then returns to the starting area and deposits into the bin before changing to the drive state and mining at the first location again. This algorithm is designed to run continuously for the allotted time and to make as many mining trips as necessary to accrue as much material as possible in the allotted time frame.

Complete Source Code Listing

```
function TestWallFollow
% -- Edited by: Emily Biller, Jake Ferguson, John Lucas, Scott Hamilton
% -- Lunabot Autonomous Navigation of Lunar Arena
% -- Drive platform simulated with iCreate Roomba w/ Hokuyo Lidar Sensor
```

```
% Add support files
addpath('./Roomba','./Hokuyo');

% Arena dimensions
arenaW = 1.66;
arenaL = 3.04;
roombaW = .33;

% State variables - startR, startL, mineR, mineL, driveR, driveL, driveM,
% returnR, returnL, returnM
state = '';
maxCount = 3; % Number of scoops before going to deposit
mineCount = 0;
currentMine=0;

% Initialize iCreate and Hokuyo
lidar = SetupLidar(4);
save('lidar');
[serPort] = RoombaInit(3);


[lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
dist = l + r;

fprintf('Rotate towards center');
while ~(((arenaL-.2) < dist) && (dist < (arenaL+.2))) || (c <1)
    dist
    arenaL-.1
    arenaL+.1
    %Turn CCW until facing middle
        fwrite(serPort, [145]); fwrite(serPort,20, 'int16'); fwrite(serPort,-
20, 'int16');
        [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
        dist = l + r;
end
fwrite(serPort, [145]);  fwrite(serPort,0, 'int16'); fwrite(serPort,0,
'int16');

fprintf('Driving to center');
while (c >(arenaW/2 +.3))
    %Drive forward
        fwrite(serPort, [145]); fwrite(serPort,30, 'int16');
fwrite(serPort,30, 'int16');
        [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
end
fwrite(serPort, [145]);  fwrite(serPort,0, 'int16'); fwrite(serPort,0,
'int16');

fprintf('Rotating to drive');
facingRight = true;
if(r>l)
    facingRight = false
end
```

```
while ~(((arenaW-.1) < dist) && (dist < (arenaW+.1))) || (c <2)
    %Turn shortest distance to face down arena
    if (facingRight)
        fwrite(serPort, [145]); fwrite(serPort,30, 'int16'); fwrite(serPort,-
30, 'int16');
        [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
        dist = l + r;
    else
        fwrite(serPort, [145]); fwrite(serPort,-30, 'int16');
fwrite(serPort,30, 'int16');
        [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
        dist = l + r;
    end
end
fwrite(serPort, [145]);  fwrite(serPort,0, 'int16'); fwrite(serPort,0,
'int16');

% Find starting position, either right box or left box
display('starting drive');
state = 'drive';
% Switch Case for state
done = 1;
while(done)
    switch state

        case 'drive'
            % Traversing the arena
            fprintf('Traversing the arena\n');
            while ~(c<.8 && c>.6)
                % Repoll sensors and calculate new distance
                [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                [rs, ls] = WallFollow(arenaW/2, rc, 'R', 'F',40);
                fwrite(serPort, [145]);  fwrite(serPort,rs, 'int16');
fwrite(serPort,ls, 'int16');
            end
            fwrite(serPort, [145]);  fwrite(serPort,0, 'int16');
fwrite(serPort,0, 'int16');

            state = 'mine';
            fprintf('State change to mineR\n');

        case 'mine'
            if(currentMine==0)
                fprintf('Mine right side\n');
                [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                center = c;
                    % Turn CW until facing corner
                fwrite(serPort, [145]);  fwrite(serPort,-20, 'int16');
fwrite(serPort,20, 'int16');
                lc
                c
                while(lc>center+.1)
                   [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                   fwrite(serPort, [145]);  fwrite(serPort,-20, 'int16');
fwrite(serPort,20, 'int16');
                end
```

```
                fwrite(serPort, [145]);  fwrite(serPort,0, 'int16');
fwrite(serPort,0, 'int16');

            elseif(currentMine==2)
                 % Turn CCW until facing corner
                fprintf('Mine left side\n');
                [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                center=c;
                fwrite(serPort, [145]);  fwrite(serPort,20, 'int16');
fwrite(serPort,-20, 'int16');
                    while((rc>center+.1))
                        [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                        fwrite(serPort, [145]);  fwrite(serPort,20, 'int16');
fwrite(serPort,-20, 'int16');
                    end
                    fwrite(serPort, [145]);  fwrite(serPort,0, 'int16');
fwrite(serPort,0, 'int16');

            else
                fprintf('Mine middle\n');
            end
            currentMine=rem(currentMine+1,3);

            while mineCount < maxCount
                while (c > .3 + (mineCount * .15))
                    %drive forward
                    fwrite(serPort, [145]);  fwrite(serPort,30, 'int16');
fwrite(serPort,30, 'int16');
                    [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                end
                mineCount=mineCount+1;
                fwrite(serPort, [145]);  fwrite(serPort,0, 'int16');
fwrite(serPort,0, 'int16');
                if(mineCount ~= maxCount) %if last run just return from where
you finish mining
                    while (c < .8)
                        %drive back
                        fwrite(serPort, [145]);  fwrite(serPort,-30,
'int16'); fwrite(serPort,-30, 'int16');
                        [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                    end
                end
                fwrite(serPort, [145]);  fwrite(serPort,0, 'int16');
fwrite(serPort,0, 'int16');
            end
            state = 'return';
            % If full

        case 'return'
            %Finding the center
            if(currentMine==0)
                currentSide = 'L';
            else
                currentSide = 'R';
            end
```

```
            fwrite(serPort, [145]);  fwrite(serPort,0, 'int16');
fwrite(serPort,0, 'int16');
            display('find bin');
            while (c < arenaL -.45)
                %drive back
                if(currentSide == 'L')
                    [rs, ls] = WallFollow(arenaW/2, lb, currentSide, 'B',
30);
                else
                    [rs, ls] = WallFollow(arenaW/2, rb, currentSide, 'B',
30);
                end
                fwrite(serPort, [145]);  fwrite(serPort,rs, 'int16');
fwrite(serPort,ls, 'int16');
                [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
            end

            fwrite(serPort, [145]);  fwrite(serPort,0, 'int16');
fwrite(serPort,0, 'int16');
            state = 'deposit';

        case 'deposit'
            fprintf('depositing\n');
             while mineCount >0

                while (c < arenaL-.45)
                %drive back
                fwrite(serPort, [145]);  fwrite(serPort,-40, 'int16');
fwrite(serPort,-40, 'int16');
                [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                end
                mineCount=mineCount-1;
                while (c > arenaL-.48)
                %drive forward
                fwrite(serPort, [145]);  fwrite(serPort,40, 'int16');
fwrite(serPort,40, 'int16');
                [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar);
                end

             end
             state = 'drive';
    end
end
DisconnectLidar(lidar);



function [rs, ls] = WallFollow(setPoint, sensor, side, dir, speed)
% -- PARAMS:
% -- setPoint, the desired distance from the wall
% -- rc, the distance from the hokoyu of the rc.
% -- side, either 'R' or 'L' depending if want to follow left wall or right
wall.
% -- dir, 'B' for backwards, 'F' for forwards
% -- RETURNS:
```

```
% -- rs, speed of right wheel
% -- ls, speed of left wheel

    if(dir == 'F')
        angleDist = sqrt(2) * setPoint;
        if (sensor > angleDist +.03)
            %Turn towards wall
            rs = speed -5;
            ls = speed +5;
        elseif( sensor < angleDist - .03)
            %Turn away from wall
            rs = speed +5;
            ls = speed -5;
        else
            %go straight
            rs = speed;
            ls = speed;
        end

        if (side == 'L')
            %switch the two speeds;
            temp = rs;
            rs = ls;
            ls = temp;
        end

    end

    if(dir == 'B')
        angleDist = (setPoint/sqrt(3)) *2;

        if (sensor > angleDist +.03)
            %Turn towards wall
            rs = speed -5;
            ls = speed +5;
        elseif(sensor < angleDist - .03)
            %Turn away from wall
            rs = speed +5;
            ls = speed -5;
        else
            %go straight
            rs = speed;
            ls = speed;
        end
        rs = -rs;
        ls = -ls;
        if (side == 'L')
            %switch the two speeds;
            temp = rs;
            rs = ls;
            ls = temp;
        end

    end
```

```
end


function [lb,l,lc,c,rc,r,rb] = poll_sensors(lidar)
[scan_data] = LidarScan(lidar);
lb = scan_data(682) / 1000;
l  = scan_data(597) / 1000;
lc = scan_data(469) / 1000;
c  = scan_data(341)/ 1000;
rc = scan_data(213) / 1000;
r  = scan_data(85) / 1000;
rb = scan_data(1) / 1000;


load lidar.mat
DisconnectLidar(lidar)
clear
```

Test Results

The robot was tested using the iRobot Create and a mining arena with dimensions of 1.66 meters by 3.04 meters. The algorithm changes depending on the dimensions of the arena, allowing for the robot to work in different areas with different dimensions. Upon testing the algorithm, everything is functioning properly including mining in 3 separate locations to avoid digging itself into a hole. The robot successfully mines until the maximum it can hold, then returns to the bin to dump what it has mined. The robot successfully traverses the area again and mines in a new location. This is repeated for the three mining locations and the robot runs on cycle of the three locations.

Safety Precautions

There really aren't user safety precautions since the user only selects run and stop to control the autonomous operation. Our main safety precautions for future groups include careful use of the iRobot and Hokuyo Range Finder since both can be very sensitive and are very expensive to replace. Some other smaller issues that we found during the testing period

involved the range finder. It can be very sensitive, which makes it impossible to be in the arena, while the run is happening. If the arena is made just for testing purposes it should have perfectly straight walls with very distinct corners, so that the range finder will run smoothly, because it can pick up the smallest of errors. Cords can also be an issue if they are in the way of the iRobot while it's traversing, so someone must hold them at all times to keep them out of the way. If the arena is made just for testing purposes.

Reflections

To reflect on our project, it went pretty smoothly, the major issues that we faced were in the debugging of the Matlab code and finding someone to let us into the lab where we could test. Since our team was nicely sized and had experience in Matlab, these problems were solved pretty quickly.

Appendix 1

Autonomous Navigation and Mining

User Manual

Emily Biller

James Ferguson

Scott Hamilton

John Lucas

Travis Williams

Instructor: Dr.Yenumula Reddy

Faculty Sponsor: Dr. Powsiri Klinkhachorn

Table of Contents

Drive State

The first state is the drive state. The drive state calls the function WallFollow until the robot gets a set distance away from the wall. The function WallFollow takes 5 parameters setPoint, sensor, side, dir and speed. The first parameter is SetPoint is the distance you would like the robot to travel from the wall, the default value is half the arena width. The second parameter is Sensor, which is the value from the sensor; the default value for sensor that is received from the hokuyo sensor is the RC (right center). The third parameter is Side, pass in 'R' for follow right wall and 'L' for follow left wall. The fourth parameter is dir, pass in 'F' to drive forward and 'B' to drive back. The final parameter is speed, this is how fast you want the robot to travel, the default value is 30. WallFollow returns two values RS, and LS these are the speeds for the right wheel and the left wheel that you send to the robot.

Mining State

The second state is Mine. At the beginning of this state the robot will either remain facing ahead or turn 45 degrees clockwise or counter clockwise depending on the current mining spot. The default is to mine right first then middle then finally mine the left side. Once the robot facing the correct direction the robot will travel forward until it is close to the wall simulating a mine. It will then repeat this the number of times the variable mineCount is set to. The default value for mineCount is 3.

Return State

The third state is Return. It is very similar to Drive except the parameters passed to WallFollow are different. If the robot is returning from mining the right side or in the middle then the robot will follow the right wall using the right back sensor. If the robot is returning from the left mining area the robot will follow the left wall using the left back sensor. Return continues to drive back until the center value

Deposit State

The final state is Deposit. In the deposit state the robot drives forward for a small set distance than drives backwards the same distance simulated shaking to empty the bucket. Deposit goes forward and back as many times as the mineCount. Every deposit the mineCount is decremented by one. Once, mineCount equals zero the state is switched to Drive.

Appendix 2

Autonomous Navigation and Mining

Maintenance Manual

Emily Biller

James Ferguson

Scott Hamilton

John Lucas

Travis Williams

Instructor: Dr.Yenumula Reddy

Faculty Sponsor: Dr. Powsiri Klinkhachorn

Maintenance of Software

The Software of the AUTOBOT System should be kept in its original form. Any alterations performed by the user of this system should be kept separate in order to assure that the original software's integrity is maintained. If the original software becomes corrupt, the best course of action is to notify the licensing company and ask for a new copy of the original software. Although this software is designed to be usable with various drive systems, certain limitations might be applied without the design team's knowledge. Therefore, it is suggested that as few changes to the software be made as possible to incorporate it into your robot.

Maintenance of iRobot

The iRobot should be kept in a clean and dry environment, it's not meant for outside use; if it becomes dirty it can be wiped down with a clean dry rag. The unit should be checked for any damage to it or the battery pack before use, any damage that is found could become worse if operated. The wheels should be checked before each use to insure that they are free of debris and any other obstructions, as to insure they will not be damaged when operated. When not in use the iRobot unit should be plugged in to keep charged. If the unit begins to malfunction or become physically broken, contact the manufacturer of the iRobot.

Maintenance of Laser Ranger Finder

The Hokuyo Laser Range Finders should be kept in a clean and dry environment, our particular range finder is not meant for outside use. The Rangefinder should be inspected for any signs of damage before use. It should be periodically wiped down with a clean dry cloth as to keep the sensor clean and insure accuracy, do not use water or any chemicals to clean the sensor. If the unit begins to malfunction or becomes physically broken then the manufacturer of the laser range finders maintenance sheet should be examined. The full parts' listing for the rangefinders is the URG-04LX-UG01 Scanning Laser Rangefinder from Hokuyo.

Maintenance of Hardware

All cables connecting the computer to the iRobot and Laser Rangefinder should be put away safely and correctly to prevent and damage or kinking of the cables in a clean and dry environment. Before use, all cables should be inspected for any cuts or damage on the cables and all connectors as to prevent damage to them and to all other hardware they are connected too. The arena should be cleared of all dirt and checked to make sure that all walls are relatively straight so as not to create false readings in the laser rangefinder.

Appendix 3

Original Design Proposal

http://lcsee.wvu.edu/seniordesign/2013fallee480-gp06/Design.pdf

Appendix 4

Summary of Changes

All of our changes were made because of the use of the iRobot instead of the mining robot. Our original documents stated that we would be working on the code to autonomously navigate the mining robot for the NASA Mining competition; we are still helping with the robot but not directly with the mining robot. The major change is that we are developing an algorithm for autonomous navigation and mining for the robot by testing on an iRobot create and coding in Matlab instead of C#. The change from our original proposal was made because of the inability to test on the mining robot. The mining robot, throughout the semester has been undergoing changes to the hardware design, which made it impossible for us to test our code. Dr. Klink decided that it was best for us to develop an algorithm for autonomous mining and navigation using the iRobot.