SUNFEST Technical Report TR-CST01DEC05, Center for Sensor Technologies, Dept of Electrical and Systems Eng, Univ. of Pennsylvania, Philadelphia, PA 2005

University of Pennsylvania

SUNFEST

NSF REU Program
Summer 2005

PATH PLANNING MOBILE ROBOTICS

NSF Summer Undergraduate Fellowship in Sensor Technologies
Louie Huang (Electrical and Systems Engineering) - University of Pennsylvania
Advisor: Dr. George Pappas

ABSTRACT

One of the fundamental problems in the field of robotics is path determination and motion planning. The project described in this paper focuses on path determination in a known environment. The goal is to enable a mobile robot to successfully navigate an environment according to a specified temporal logic formula. On a high level, temporal logic formulas can effectively provide the robot with directions on where and when to go. As part of this project, a program will be developed to formulate a continuous path plan that will fulfill the temporal logic formula supplied for the robot to follow.

In this project, the ActivMedia Pioneer 3-DX robot will be used. This robot model was chosen because it is preconfigured for basic navigation. Preprogrammed with algorithms for shortest path determination, obstacle avoidance, and localization, the Pioneer 3-DX is also capable of new navigation techniques that can be programmed in C and C++. Maps of known environments can be generated by the Pioneer 3-DX. A graphical user interface program will utilize these maps in conjunction with user supplied directions as expressed by a temporal logic formula to construct a path plan. The path determined by the program can then be relayed back to the robot for implementation.

Table of Contents

1. INTRODUCTION

In the constantly evolving field of robotics, path determination is a topic that attracts much interest because of its numerous potential applications. A robot which can plan its own path when given destinations and certain guidelines can be used for patrol and mobile surveillance or transport and delivery of items. Using robots for such applications can not only offer convenience to users but can also save lives when employed in military situations. Such beneficial uses validate the importance of research pertaining to path determination.

The project that this paper is based upon focuses on determining a path that fulfills a specified direction represented by a temporal logic formula. It stems from an earlier paper [1] published by two graduate students -- Georgios Fainekos and Hadas Kress-Gazit -- and Professor George Pappas of the General Robots Automation Sensing Perception (GRASP) Laboratory at the University of Pennsylvania. The paper presents the method by which continuous path plans can be generated for a robot in a known environment. The continuous path will be implemented by the robot and should satisfy temporal logic input. The overall goal behind this study is to allow users to effectively direct a robot on a high communication level that is close to natural human language in the form of temporal logic [1].

The sections that follow this introduction will further explain the details of the project and the progress made to date. Section 2 will discuss the background of the development of a continuous navigation path. Section 3 will introduce the ActivMedia Pioneer 3-DX, the specific robot that will be used in this project. This section will also present some key navigation behaviors that the Pioneer 3-DX is capable of and discuss how new behaviors can be programmed. Section 4 will cover the purpose and the development of the robot map graphical user interface (GUI) program, which will be used to ultimately generate paths for the robot. In section 5, discussions and conclusions of the overall project are presented. Section 6 covers recommendations for future work and Section 7 is dedicated to acknowledgements. All references can be found in Section 8.

2. OVERVIEW OF CONTINUOUS PATH CREATION

The following description of continuous path creation was originally presented and discussed in the research paper [1] that served as a foundation for this project. In the interest of brevity, the prior work on continuous path creation will be briefly summarized here. Readers interested in further detail should consult the original source.

The process of creating a continuous path for implementation begins with navigation directions. Directions are to be given in the form of temporal logic formulas. Temporal logic formulas can delineate multiple destinations and specify when the destinations should be reached. For example, an instance of a temporal logic formula can be used to direct the robot to visit all rooms but not to visit a certain room until all other rooms have been visited. Temporal logic formulas are close to human language, providing greater accessibility to users who would therefore not need to be concerned with the lower implementation of the robot's movements. A direction alone is not sufficient for path generation. The robot must know its environment through a map of the environment that pertains to the directions provided.

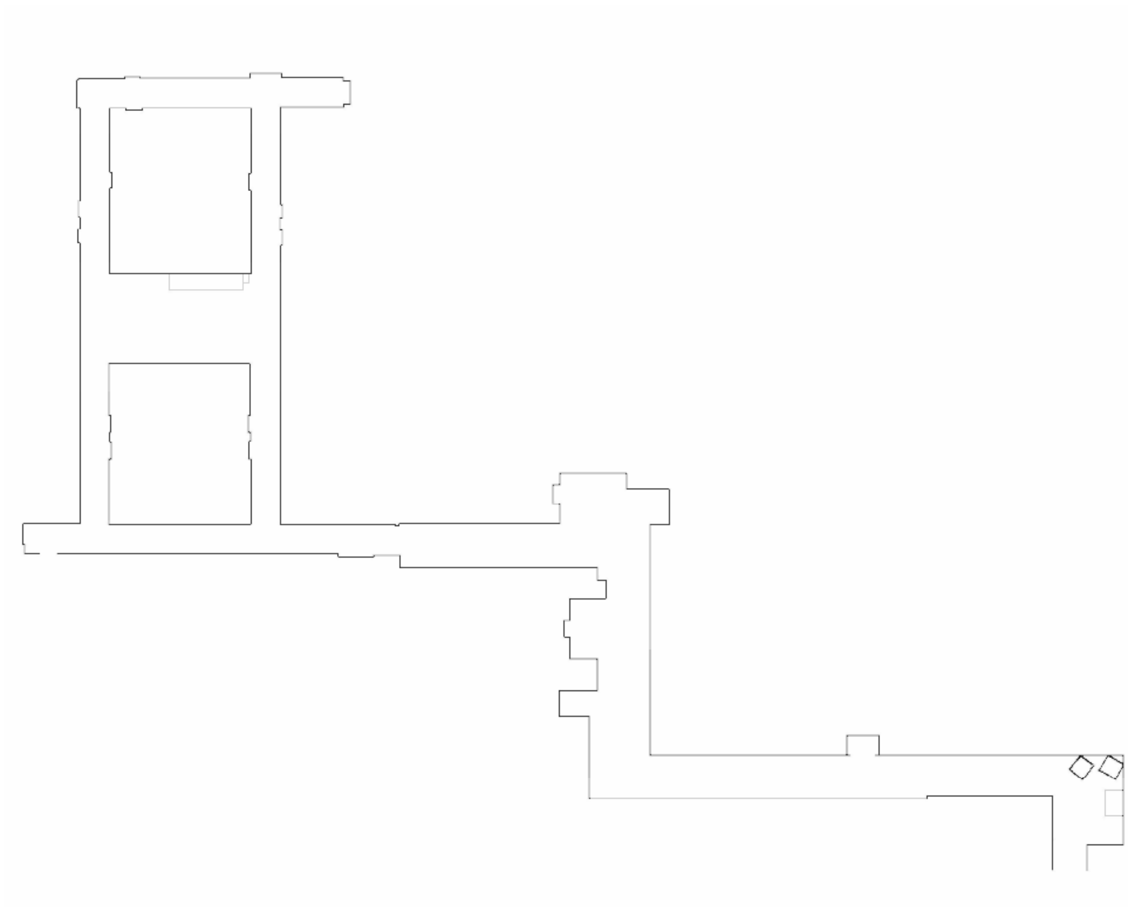


Figure 1: 2-D environment map of a floor in the Levine Building

Another necessary component for creating a path is a 2-D graphical map (as illustrated in Figure 1) delineating, from a bird's eye view, walls and obstructions of the environment to be navigated. The next step in arriving at a continuous path is to develop a discrete path through model checkers. To develop a discrete path, the 2-D graphical map needs to be partitioned into discrete units. Although other partitioning methods could also be considered, triangulation has been recommended because of its relative ease in computation and readily available algorithms. The destinations then specified by the temporal logic formula will each be composed of at least one triangle. A discrete path can be created that visits the necessary triangles to fulfill the temporal logic formula. Upon generation of a discrete path, a continuous path can be developed. The continuous path must obviously still satisfy the original temporal logic formula. For further detail on the process of continuous path creation, please refer to the original path planning paper [1].

3. ROBOT DESCRIPTION

3.1 Pioneer 3-DX

The Pioneer 3-DX robot made by ActivMedia Robotics is an all purpose robot suitable for research [2]. The robot is readily capable of basic navigation functions. One of the most important features of the Pioneer 3-DX is its ability to localize itself fairly accurately within a known environment. In Figure 2, the Pioneer 3-DX is observed from its front. Localization is made possible by the eight sonar shown in the figure; two of the sonar are hidden from view as they are mounted on the sides of the robot. The laser rangefinder that sits on top of the robot also assists with localization. The laser rangefinder has the advantage of greater range and accuracy than the sonar. However, the sonar is necessary to detect low lying obstacles close to the ground.

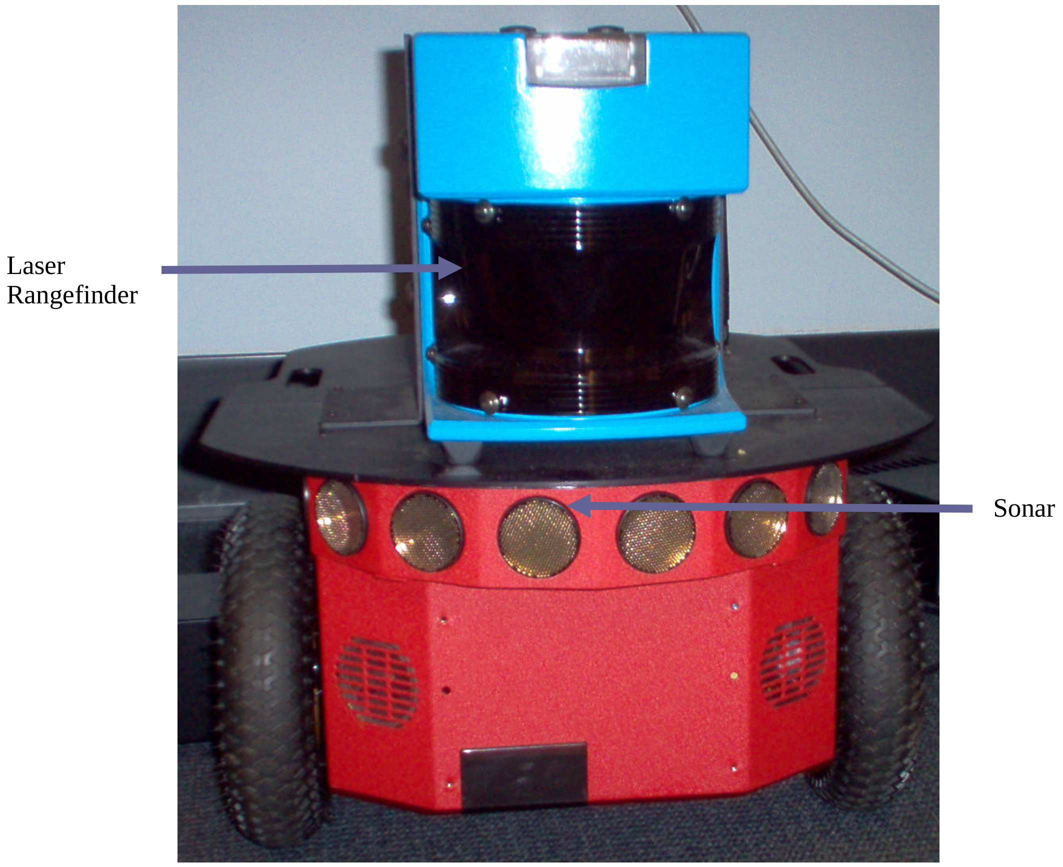


Figure 2: Front View of Pioneer 3-DX

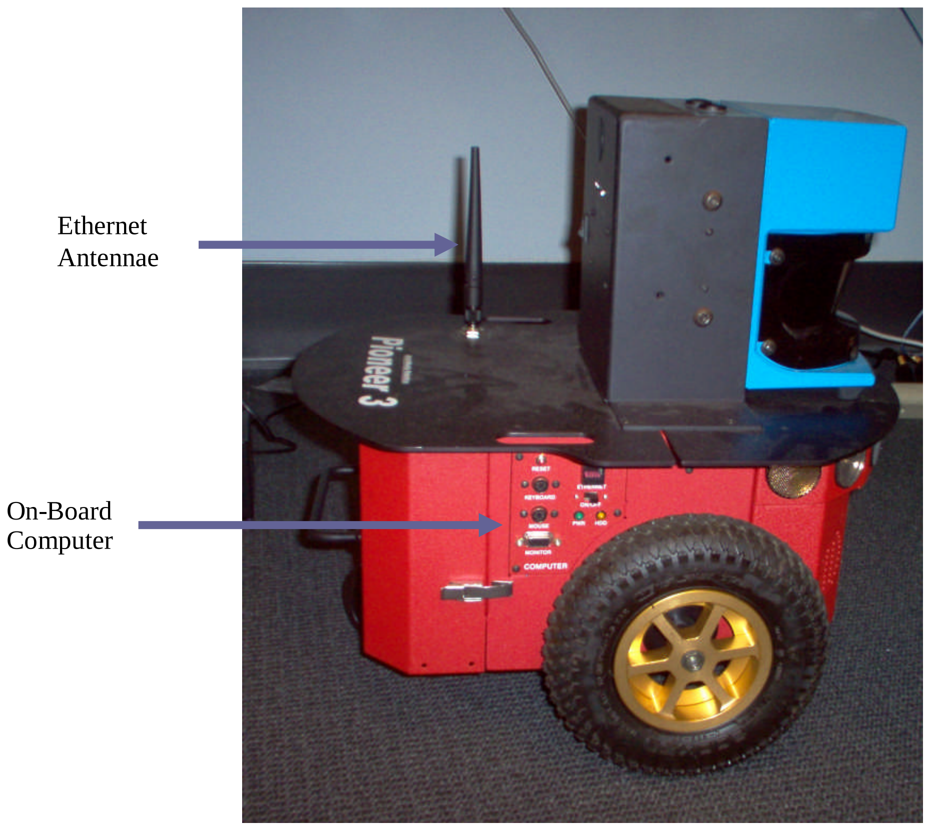


Figure 3: Side View of Pioneer 3-DX

For the Pioneer 3-DX to localize itself, it would need to know its environment. To acquaint the robot with an unfamiliar environment, a joystick can be plugged into a USB port on the robot. With the laser rangefinder activated, the robot can be driven manually with the joystick until the new environment has been fully covered. The robot has an on-board computer (see Figure 3) which can store points scanned from the laser rangefinder and create a map file similar to that shown in Figure 1. Sometimes, erroneous points can be introduced during the mapping process. Because the environment is unlikely to be static, movement by people or other mobile objects within range of the laser will be picked up and plotted on the map. The laser also lacks the ability to successfully detect transparent surfaces such as glass on windows as obstructions. See Figure 4 for the original floor map of the Levine Building. Hazy or noisy areas exist where erroneous or unintended data points are plotted. To correct these possible errors, the map file needs to be edited.

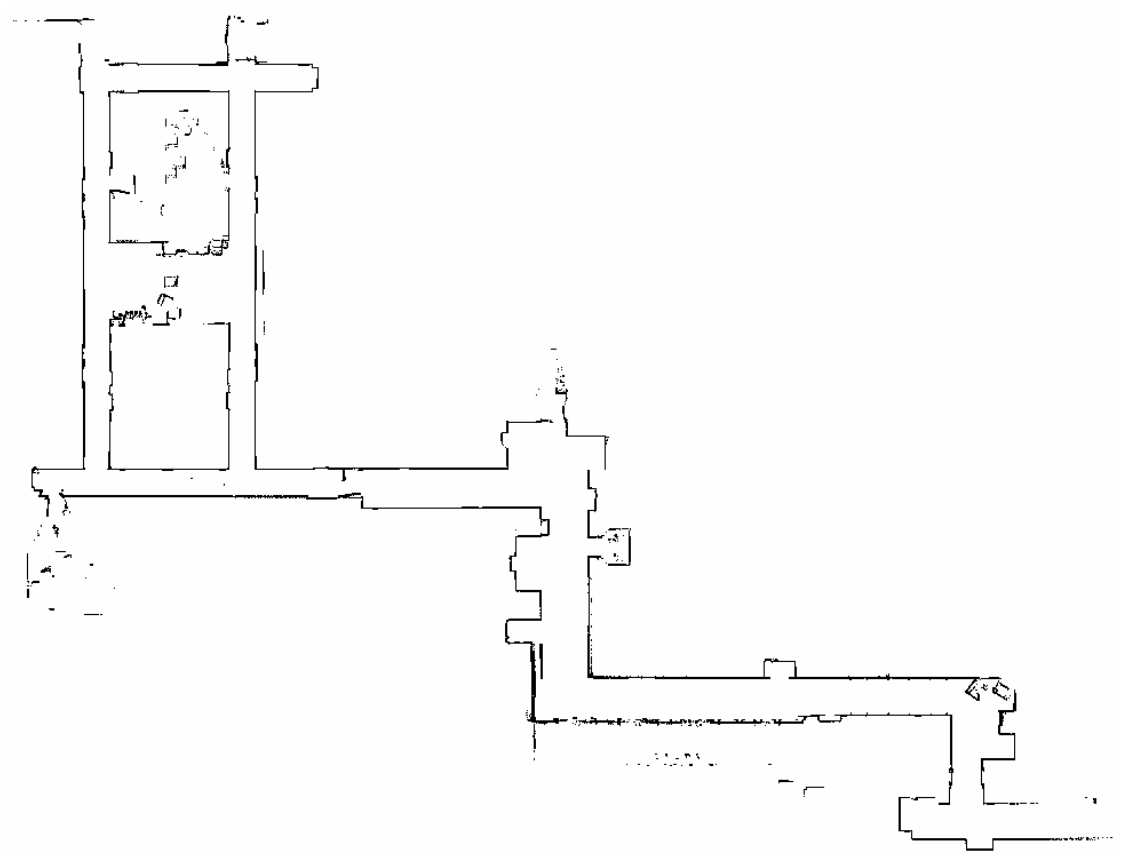


Figure 4: Robot generated map of floor in Levine Building before editing

The on-board computer of the Pioneer 3-DX is sufficient for implementing the programs required for the robot's navigation. However, due to resource restrictions, the map editing process will be fairly slow on the on-board computer. To modify the map file more efficiently, files on the Pioneer 3-DX can be sent to a base computer via wireless Ethernet. Figure 3 shows the Ethernet antennae that can wirelessly transmit information packets to another PC. Upon transferring the map, editing can be done on the base computer and then the map can be sent back to the robot for storage and future use. Besides editing out erroneous points, the base computer will also eventually be used to implement the process by which the map is partitioned and a continuous path based on temporal logic formulas supplied is created.

3.2 Robot Behaviors

The Pioneer 3-DX comes equipped with some useful high level navigational behaviors. Two of the most important behaviors are obstacle avoidance and shortest path determination. In obstacle avoidance, the robot takes readings from its laser rangefinder and sonar to determine whether there are objects blocking its path. Upon detecting an obstacle in its path it will stop to avoid collision. There are several behaviors that can follow the stop action after obstacle detection. One consequent behavior that occurs after detecting an obstruction instructs the robot to back off after stopping and turn towards a different direction. The other important behavior with which the Pioneer 3-DX is preprogrammed is shortest path determination in a known environment. This behavior requires a map file of the environment and a home position on the map from where it will start. A destination is also specified on the map. With this map file, the robot has the ability to determine a shortest path while avoiding walls specified by the map.

The robot behaviors that the Pioneer 3-DX can exhibit are developed with the ActivMedia Robotics Interface Application (ARIA). ARIA is programmed in C++ and its classes can be used to obtain readings from the robot's sensors (sonar, laser, wheel movements, etc.) [2].The robot's basic movements -- as represented by direction -- and speed can also be manipulated by ARIA. To develop new behaviors or modify existing navigation behaviors, ARIA's classes can be used in a C++ programming environment such as Microsoft Visual Studio C++. Because the on-board computer has limited resources, a programming environment such as Visual Studio would be operated at a base computer. A behavior after compilation then can be relayed back to the robot for implementation through wireless Ethernet.

Using SRI International's Saphira is often an easier way of implementing behaviors. Saphira is built utilizing ARIA and exists as a higher level programming environment for robot behaviors. As a consequence of its higher level nature, programming in Saphira is inherently simpler than programming in C++ with ARIA. Saphira includes its own programming language, Colbert, which is based on the C programming language [3]. Colbert can be used to construct activities which provide navigational instructions to the robot upon implementation. Activities can be used to provide direct movement commands which, for instance can tell the robot to move forward 10 meters and turn 90 degrees. An activity can also be used to implement robot behaviors like obstacle avoidance by producing movement commands that depend on sensor readings [3].

Saphira also provides a simulator from which activities can be interactively tested and modified. The robot used in the simulation can be an actual Pioneer 3-DX or a virtual robot that emulates actual physical robot limits. The simulator allows multiple activities to be implemented. Of course, some activities may contradict others at times in which one activity is directing the robot forward and another is simultaneously directing it to move in reverse. To resolve such movement conflicts, Saphira uses a system by which each activity is given a priority. Direct motion directions are usually given precedence over directions derived from sensor dependent behaviors [3]. For implementation of activities that involve a known environment, map files can be imported into the Saphira simulator.

4. GRAPHICAL USER INTERFACE FOR ROBOT MAPS

4.1 Purpose

As discussed earlier, map files generated using the robot's laser rangefinder may have many points or representations of obstacles which should not be there. Also, walls that really exist may occasionally fail to appear if the wall is transparent. To correct these problems, a graphical user interface is needed to modify the map file. The map can be readily accessed from a base computer that receives the file from the robot's wireless Ethernet transmission. ActivMedia, the manufacturer of the robot already provides software to edit the map. The ActivMedia Mapper 3 program allows users to eliminate erroneous points on the map and insert lines that represent walls. It also allows users to add goal points or destinations. Other additions that can be made to the map file include forbidden regions and forbidden lines, which may not be actual obstructions in the environment, but are nevertheless areas that the user wants the robot to avoid.

The Mapper 3 is fairly useful but for the purposes of this project, it is not sufficient. To fulfill the necessary specifications that would allow for the creation of the continuous path, the Mapper 3 would need to be able to partition the map into discrete regions through triangulation or some other partitioning method. In addition, the Mapper 3 would also need to be able to accept temporal logic formulas and construct a discrete path that would ultimately lead to the continuous path for the robot to implement. Because the Mapper 3 is not open source, it was decided that the best option would be to build a graphical user interface from scratch that -- in addition to including most of the Mapper 3's features -- would also allow for triangulation and path determination based on directions given by temporal logic formulas. A crucial development requirement is that the modifications made to any maps through this robot map GUI must still maintain the map files' compatibility with the robot.

The base computer that is used for the Pioneer 3-DX in this project is an IBM compatible machine that runs Microsoft Windows. To build the GUI as a Windows application, Microsoft Visual Studio .Net is used as the programming environment. C++ is used as the programming language in Visual Studio .Net to construct the GUI as a Windows form application. The choice to use Microsoft Visual Studio .Net rather than other environments is based on the ease of use that Visual Studio .Net offers for programming GUIs using the Windows operating system. The classes provided by Visual Studio .Net particular to Windows GUI programs simplify the GUI programming significantly.

4.2 Development of the GUI

The first function to be developed for the GUI was to display map files correctly. Map files have the extension .map and can be accessed from a basic text editor. A map file generated by the Pioneer 3-DX always begins with the line "2D-Map". The three lines that follow this initial line pertain to obstacle points. Most environments would not have obstacle points since walls are represented by lines. An obstacle point is most likely noise since an obstacle point represents an object with an area of 1mm^2. The three lines that pertain to any existing obstacle points provide the minimum x and y coordinates and maximum x and y coordinates in millimeters, as well as the number of obstacle points in the map. The next three lines contain similar information for obstacle lines. Following the information on obstacle lines, the map file contains the definitions for forbidden areas, forbidden lines, goals, and home points in that order. Each line contains an individual definition represented by coordinates in millimeters and begins with the word "Cairn:" and the type of object defined. After definitions of these objects, the map file contains a list of lines each of which consists of 4 numbers and represents two coordinates in millimeters. At the beginning of this list is the word “LINES” to signify that the list consists of definitions for obstacle lines in the map. After this list is the list for obstacle points which begins with the word “DATA”. This list contains lines of 2 numbers, each pair belonging to the coordinate pair of one obstacle point.

To properly display the map file, the GUI must read the map file and utilize the information provided to construct a graphical map. Extracting the data and drawing the map lines and objects in the GUI’s main panel did not prove to be a complicated task. However, the map’s coordinate system is inherently different from that used in Windows forms programming. This can be seen in Figure 5. The + sign signifies the direction in which coordinates increase positively. The map therefore needed to be adjusted according to its maximum and minimum coordinates as they are given at the beginning of each map file, flipping them vertically due to the inverting nature of the y axis in the Windows forms coordinate system. For details, please refer to the translate function in the Form1.h class in the Appendix. Another problem arose in the display of the map because the coordinates given in millimeters were too great in magnitude to be displayed at a level for the user to view the map as a whole. Therefore, the x and y parts of each coordinate need to be reduced in magnitude through dividing by a reduction factor. The standard reduction factor when a map is loaded is 50. This number was determined through trial and error to produce a suitable viewing size for the map.

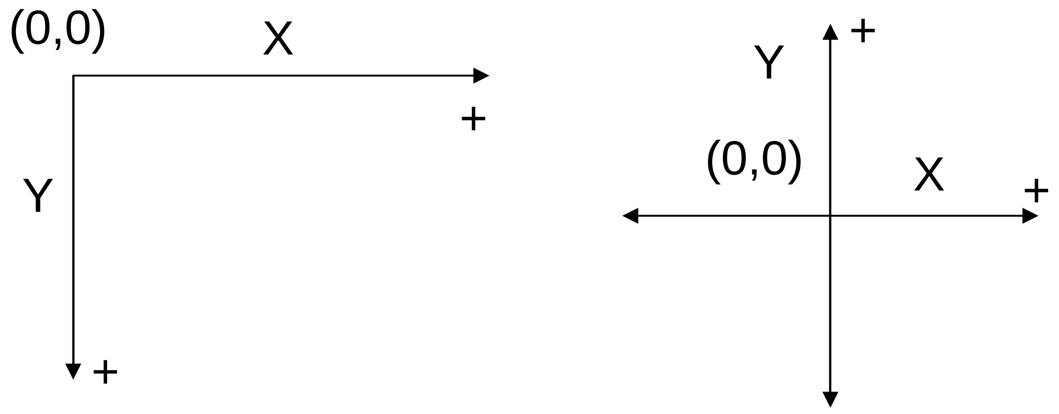


Figure 5: Differing coordinate systems: Windows form coordinate system (left) and standard map coordinate system (right).

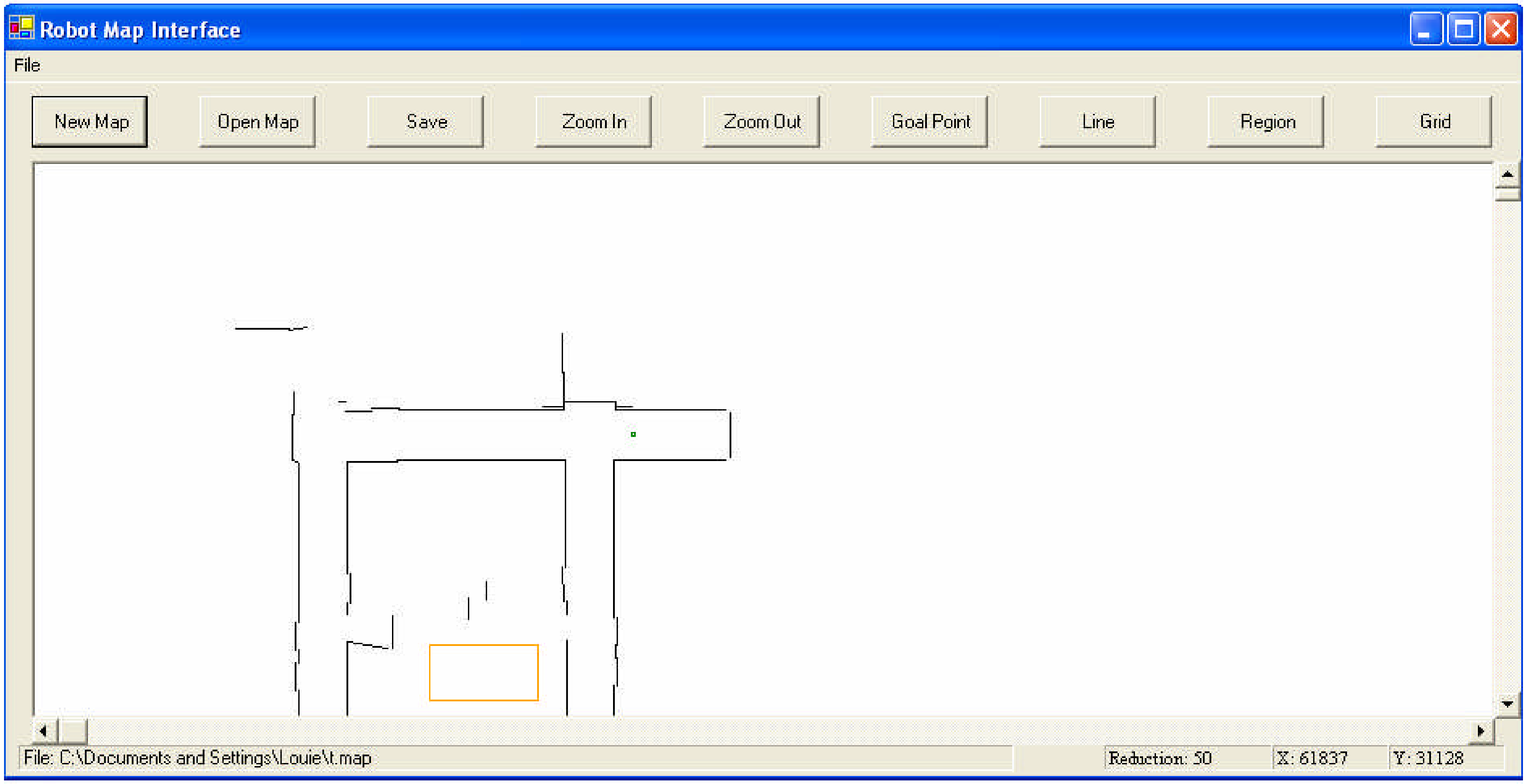
Even with the reduction factor at 50, the main panel is too small to display every map. The map shown in Figure 1 of a floor in the Levine Building could not be fully shown in the panel. To account for this problem, scrollbars were added to the GUI. The vertical and horizontal scrollbars, whose limits are determined by the maps’ maximum coordinates, successfully allow any map to be fully displayed. To allow for greater viewing versatility, in addition to the scrollbars, two buttons were added to the GUI that would control zoom. One button would zoom in on the image and essentially magnify it by reducing the reduction factor. The second button would zoom out and essentially shrink the image by increasing the reduction factor. With the addition of the zoom functions, most maps can be scrutinized up close or viewed in their entirety without scrolling on the main panel.

The next functions to be implemented would allow for the additions of lines, goal points, and regions to a map file. Three buttons are added to the GUI each to represent a type of object to be drawn. For example, if the line button is pressed, the map file will only accept the additions of lines. The reason these buttons are necessary is because each object is drawn similarly using mouse clicks. Lines are to be drawn by clicking down the mouse button at the site where the first endpoint should be. The mouse button is held down until the mouse is moved to the second endpoint. Goals are drawn by clicking any point in the map. For ease of viewing, goals are shown as small green squares. The center of each square is the actual goal point. Regions are currently general custom structures added to the map. They may constitute future forbidden regions but they can also be used to parameterize the map. Though triangulation is likely to be the parameterization method to be used, the regions' function is currently programmed to construct any polygon. To define a simple rectangle, the user only needs to click two points, which will always be the upper left and the lower right vertices. The second point must be double clicked to signify that the region will be a rectangle. Any other type of polygon can be constructed by mouse clicks. Each mouse click will define a vertex and the last vertex will require a double click signifying the end of the region definition.

The added lines, goal points, and regions are not saved to any files when drawn onto a map. To register the additions made to the map, a save function was implemented in the GUI. To maintain the integrity of the original map file and its compatibility with the robot, the definition for each new line and goal must be added to the file in the appropriate areas with the appropriate syntax. The regions defined in the GUI are not necessarily forbidden regions and therefore their definitions are stored in an auxiliary text file with a filename that is always the name of the original map followed by "_regions". In addition to saving the map files generated, the GUI will also save newly created map files. The new map function in the GUI will generate a blank map with user defined dimensions. Maps newly created from the GUI will be indistinguishable in format from robot generated maps.

To date, the last function to be added to the GUI is the grid display and the position tracker. When editing a map, it is useful to have a grid and to know the coordinates of the region being modified. The grid function can be displayed when a map file is opened by a click of the grid button. It can also be made invisible by a second click. Grid lines are 1000 mm or 1 meter away from one another. As a result, the perceived distances shrink or grow when a zoom is applied to the map. The position tracker simply tracks the mouse position over the main panel where the map is displayed. Whenever the mouse moves, the tracker display located at the bottom right corner of the GUI is updated. The coordinates are displayed in millimeters and properly match the coordinate system native to the maps as portrayed in Figure 5.

The latest robot map GUI is shown in Figure 6. Each of the 9 buttons represents a function described. The green square represents a goal point while the orange rectangle is a region. The black lines represent walls in the map. The bottom of the window shows the file currently being accessed. At the bottom right corner, the reduction factor is displayed along with the current mouse position.



File: C:\Documents and Settings\Louie\t.map

Reduction: 50

X: 61837

Y: 31128

Figure 6: Snapshot of the robot map GUI; the map file t.map is being loaded at a reduction factor of 50

5. DISCUSSION AND CONCLUSIONS

The goal of the GUI in this project was to successfully access robot generated map files and make modifications. In this aspect, the GUI in its current stage is fairly successful. With the GUI, new lines and goals can be added to existing maps or created in new maps. Robot compatibility with files modified and created by the GUI has been successfully maintained. Regions which may be later used to partition the environment can be added in the GUI. As the regions defined in the GUI are not directly useful to the robot, regions made to a map file are stored in separate text file with a similar name as the original map. The GUI, unlike the Mapper 3 offered by ActivMedia, does not allow users to add home points or forbidden areas or lines. For the project at hand, these features do not appear to be needed. However, if this should change in the future, simple modifications can be made to the robot map GUI program to allow for the addition of home points, and forbidden regions and lines.

One peculiarity of the GUI is that sometimes at large reduction factors, the coordinate display may show coordinates slightly different from their expected appearance. The scale of map coordinates stored is 1 mm per pixel on a monitor display. This scale can only display a small piece of a map at any one time. To increase the distance per pixel so that more of the map can be viewed at once on the screen without having to scroll around, a reduction factor is maintained by the GUI. The reduction factor reduces the magnitude of all coordinates in the map. For higher calculation speeds, the division of coordinates by the reduction factor is made an integer division, dropping the decimals. This rounding off results in slightly inaccurate displays in coordinates of up to a 10 mm deviation under a reduction factor of 50. However, this slight inaccuracy is insignificant because 10 mm is relatively small in any normal sized environment. Also, if accuracy is needed, the zoom function can focus in on any area of a map. Though accurate work can be done by zooming in or lowering the reduction factor, the smaller magnified portion of the map will make editing work more difficult.

One key feature which has not been implemented in the GUI is an erase function. Currently, cleaning up erroneous data points and lines requires the use of Mapper 3. The erase function is fairly important as the GUI will eventually need to function independent of the Mapper 3 in modifying maps. The only remedy for erasing lines or goals that were inadvertently added is by reopening the map in the Mapper 3 and using its erase function. An alternative method is to open the map file in a text editor. This method is painstaking, however, as the user must know at least how many lines were added to the map file. The most recent lines added are always added to the beginning of the list by the mapping GUI. Currently, the only method by which added regions can be removed from a map is to edit the region file with a text editor. A region file begins with a line that identifies the number of regions in the map, followed by region definitions that are individually identified by a number representing the order in which each was created. Each region definition also identifies the number of points in the region and all the points that pertain to the region. In order to be able to delete an erroneous region, a user would have to know either the defining characteristics of a region as determined by number of points and point location, or when the region was added in reference to all other regions.

6. RECOMMENDATIONS

The most pressing focus for future research should be the creation of an erase function. Besides providing independence from the Mapper 3, the erase function is also crucial to deleting erroneous regions generated by the mapping GUI. These regions are not accessible from Mapper 3 and currently can only be deleted through a text editor. File verification safeguards are another feature that could be added to enhance the robustness of the current GUI. Currently, when opening a file, the open file dialog filter allows only files with .map extensions to be accessed. However, the program does not go further in verifying that the actual map file is valid. If the format is invalid or the map file has been tampered with, the program in its current state will just crash. Within the map file, the maximum and minimum points which determine the size of the map to be displayed need to be verified each time a map file is opened and corrected if they are inconsistent with data. Occasionally, the maximum and minimum listed in the map file are not correct. This results in some lines not being displayed in the GUI because the lines are out of the range of the display created for the map.

The current version of the GUI is clearly not complete in the sense that it does not accept temporal logic formula input nor does it partition the environment. Without these functions, the mapping GUI is not substantially more enhanced in features than the Mapper 3. The ability to define general regions is a step towards environment partitioning since this feature can be integrated into a future partitioning function. The current region creator can generate triangles and this will be useful since triangulation will likely be the partitioning method. Upon successful implementation of map partitioning and temporal logic formula processing in the GUI, further work in regards to path plan creation and implementation will require working with ARIA classes. ARIA classes can be used to implement the robot’s motion in a continuous path that is developed by the GUI. Saphira and Colbert will not be used for two reasons. The first is that ActivMedia no longer supports Saphira and will not offer future updated versions. The second is that ARIA, being on a lower programming level, will provide more detailed control in implementation.

7. ACKNOWLEDGMENTS

This project has provided an invaluable opportunity to conduct research in the developing field of robotics. The author has gained a great amount of knowledge in robotics and programming from this project. In this sense, this experience has been truly rewarding. The author is also grateful to have been given the privilege to work on such a fascinating project.

I would like to thank Professor George Pappas for entrusting this project to me for the summer and providing valuable guidance and advice. I very much appreciate having been given the opportunity to work on this project. I would also like to thank Hadas Kress-

Gazit, the graduate student responsible for the project, for working closely with me throughout the summer and offering her knowledge and support.

I would like to thank Professor Jan Van der Spiegel for allowing me to be part of the SUNFEST program. The program has provided me with valuable knowledge in regards to technical writing, presentations, conducting research, and engineering ethics. Finally, I would like to thank the National Science Foundation for its support of Research Experience for Undergraduate Programs like SUNFEST at the University of Pennsylvania.

8. REFERENCES

[1] G.E. Fainekos, H. Kress-Gazit, G. J. Pappas, Temporal Logic Motion Planning for Mobile Robots, IEEE Conference on Robotics and Automation, Barcelona, Spain, April 2005.

[2] Mobile Robots, ActivMedia Robotics, LLC, 2005, www.mobilerobots.com

[3] K.G. Konolidge, Saphira Software Manual, SRI International, Menlo Park, California, 2001.