Programming an E-Puck Robot to Create Maps of Virtual and Physical Environments

Pablo Tarquino1, Kevin Nickels1,

1 Trinity University Engineering Department, San Antonio, Texas

ptarquin@trinity.edu

knickels@trinity.edu

**Abstract.** This project is a first step towards research on implementing Simultaneous Localization and Mapping (SLAM) techniques in robots. The paper provides theoretical background for SLAM and occupancy grids, which are used in the project to create maps. The paper also describes the software, Python and Player/Stage, and the hardware, the E-Puck robot, used in the project. This project successfully programmed an E-Puck robot to map an unknown virtual and physical environment. The virtual environment has perfect localization conditions, while the physical environment has error in its localization. A comparison of these maps shows that the map of the virtual environment is highly accurate, while the map of the physical environment is less accurate due to odometery errors.

**Keywords:** Occupancy Grid, Simultaneous Localization and Mapping, E-Puck Robot, Table-Top Robot, Robot Programming

1 Introduction

Imagine the task set before the next-generation vacuum-cleaning robot - to vacuum around an unknown environment, clean the floor as it goes, and make a map not only of the rooms but of the dirty spots in the environment.  In order to do this, it will need to create a map of the house, and to figure out where it is within this (incomplete) map at the same time.

This project focused on programming a simulated and physical E-Puck robot to create a map of an unknown environment. The project used the popular Player/Stage software package. Player [1] is a robot device interface used to communicate with the E-Puck robots. Stage [2] is a simulator used for testing of the code. The computer language Python was used to implement the mapping behaviors of the robot. In addition, the graphic Python library Matplotlib was used to visually represent the map the robot constructed.

This project is an initial step towards researching Simultaneous Localization and Mapping (SLAM). This project explored only the mapping aspect a robot might encounter within SLAM. For this work, perfect conditions were assumed, meaning that the robot had a perfect knowledge of its position at every point within the map. With this assumption, the robot observed its surroundings via infrared proximity sensors and constructed a map of the surroundings of the robot.

This paper is organized as follows. Section 2 provides a general overview of SLAM theory. Section 3 explains the software used in the project: the programming language Python and the software package Player/Stage. Section 4 describes the components of the E-Puck robot. Section 5 gives the basic theory of occupancy grids and shows how they were used within the project. Section 6 provides the experimental setup of the project, which includes an explanation of the mapping technique used and the virtual and physical environment created in the lab. Section 7 makes a comparison between the virtual and physical maps created. Finally, Section 8 discusses the results and conclusions obtained from the project.

2 Simultaneous Localization and Mapping

Localization is the process of estimating the position of a robot within a known environment, while mapping is the process of creating a model of the robot’s environment [13].

The Simultaneous Localization and Mapping (SLAM) problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map [3]. In this manner, the problem of SLAM can be seen as a “chicken and egg” problem: A robot needs an accurate map to know its location within the environment, yet a precise knowledge of the location within an environment is needed to create an accurate map [4].

SLAM is a process by which a mobile robot can build a map of an environment and at the same time use this map to deduce its location. In SLAM both the trajectory of the platform and the location of all landmarks are estimated on-line without the need for any *a priori* knowledge of location [3].

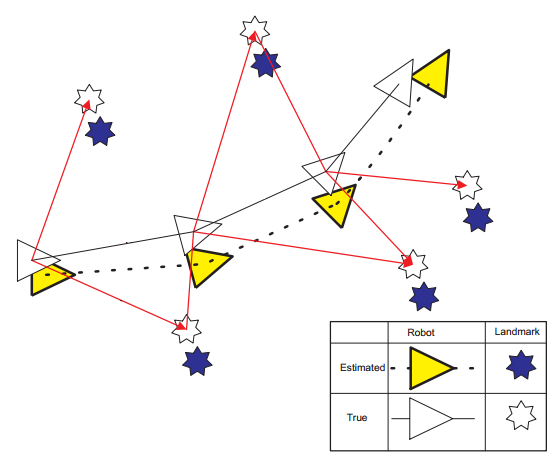


Fig. 1. The true and estimated location of a robot and landmarks as the robot moves (from [3])

Consider the robot shown in Figure 1. The robot has sensors to perceive the landmarks as well as sensors to keep track of its location. Commonly used sensors that perceive landmarks include cameras, sonar, laser, or infrared technology. Commonly used sensors that keep track of location include odometers or Global Positioning System (GPS) technology [3][15].

As the robot moves, the estimated robot and landmark locations become distinctly different from the true robot and landmark locations. This is because the sensors are subject to error, often referred to as measurement noise. This error is statistically dependent, which means that the error accumulates over time and affects all future measurements. Figure 10 in Section 7 depicts a map made with accumulated error [15].

It is the task of SLAM to obtain more accurate estimates of the true location of the robot and the landmarks. This is accomplished by using probabilistic techniques to filter out the noise found in the measurements, primarily by re-observing features seen previously (with lower uncertainty). Probabilistic algorithms approach the problem by explicitly modeling different sources of noise and their effects on the measurements [3][15].

SLAM also aids in solving the data association problem. The data association problem is the problem of determining if sensor measurements taken at different points in time correspond to the same physical object in the world. For example, a robot may complete a cycle inside an environment while mapping. When closing the cycle, the robot has to ﬁnd out where it is relative to its previously built map. A robot that successfully localizes itself while closing the cycle reduces the uncertainty associated with that landmark, helping to solve the data association problem. Once this is accomplished, the robot can map multiple cycles around the environment. Though the sensor data accumulates error, using probabilistic methods allows the data to be filtered, creating accurate estimates of the environment [15].

Techniques to implement SLAM include methods such as a Kalman filter or a Rao-Blackwellised filter. The project has not yet used these probabilistic methods. An in-depth overview of these and more methods can be found in [3].

3 Software

This section explains the software used throughout the project. The robot was programmed using the user-friendly Python. When simulating, the python controller interfaced with the Player/Stage software package, and when running on the E-Puck, the same controller was interfaced with the Player package.

3.1 Python

Python was used as the primary programming language throughout the project. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics [5] [6].

Two characteristics made Python ideal for the project. In the first place, Python is a language that can be supported with the freeware Player, which was extensively used throughout the project. Second, Python is easy to learn due to its simple syntax and extensive online support, furthering the project objective of improving access to robotics research for undergraduate students.

3.2 The Player/Stage Project

The *Player/Stage Project* provides open-source tools that simplify controller development, particularly for multiple-robot and sensor network systems. The project offers a combination of flexibility and speed that makes it one of the most useful and popular robot development environments available [4].

Player is a socket-based device server that allows control of a wide variety of robotic sensors and actuators. Player executes on a machine that is physically connected to a collection of such devices and offers a TCP socket interface to clients that wish to control them. Clients connect to Player and communicate with the devices by exchanging messages with Player over a TCP socket. Because Player’s external interface is simply a TCP socket, client programs can be written in any programming language that provides socket support, such as C, C++, Tcl, Python, Java, and Common LISP [1].

Stage can simulate a population of mobile robots, sensors and environmental objects. Its purposes are to enable rapid development of controllers that will eventually drive real robots, enable robot experiments without access to the real hardware and environments and to enable experiments with devices that do not exist yet. Stage enables experiments with large populations of robots that would be prohibitively expensive to buy and maintain [1].

4 E-Puck Robot

The project used an E-Puck robot due to the affordability and ease of use that this table-top robot provides. The E-Puck robot was developed by the Swiss Federal Institute of Technology in Lausanne (EPFL) for teaching purposes and is currently a commercial product available from Cyberbotics Ltd [14][7].

E-Puck robots are small, having a diameter of 70 mm, height of 55 mm, and a weight of 150 g. Apart from its small size, the robot’s dual motors, proximity sensors, wireless communication, and programmability made it ideal for this project. The robot has two stepper motors with a 50:1 reduction gear. Each of these motors can be separately programmed to turn and each motor keeps track of the displacement of the robot with integrated odometers. The robot has eight infrared sensors that measure ambient light and proximity of objects up to 6 cm. The robot supports computer-robot or robot-robot wireless communication via Bluetooth [7]. Additionally, the robot is easy to program and compatible with popular robotic software, such as the commercial Webots or the open-license Player/Stage project.

A Player Driver for the E-Puck [8] was modified for the current version of Player (3.0.2), and integrated with the demonstration programs distributed with the E-Puck [9]. This player driver runs on the E-Puck robot, taking Bluetooth commands from the Player server and returning sensor data to the Player server.

The E-Puck’s major components can be seen in Figure 2. A full list of the E-Puck’s specifications can be found in [7].

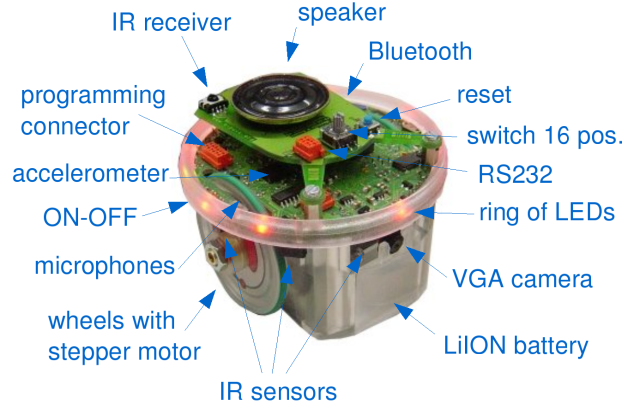
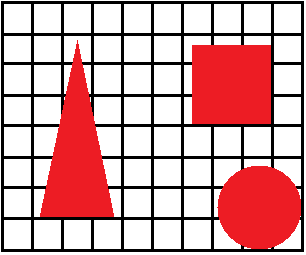


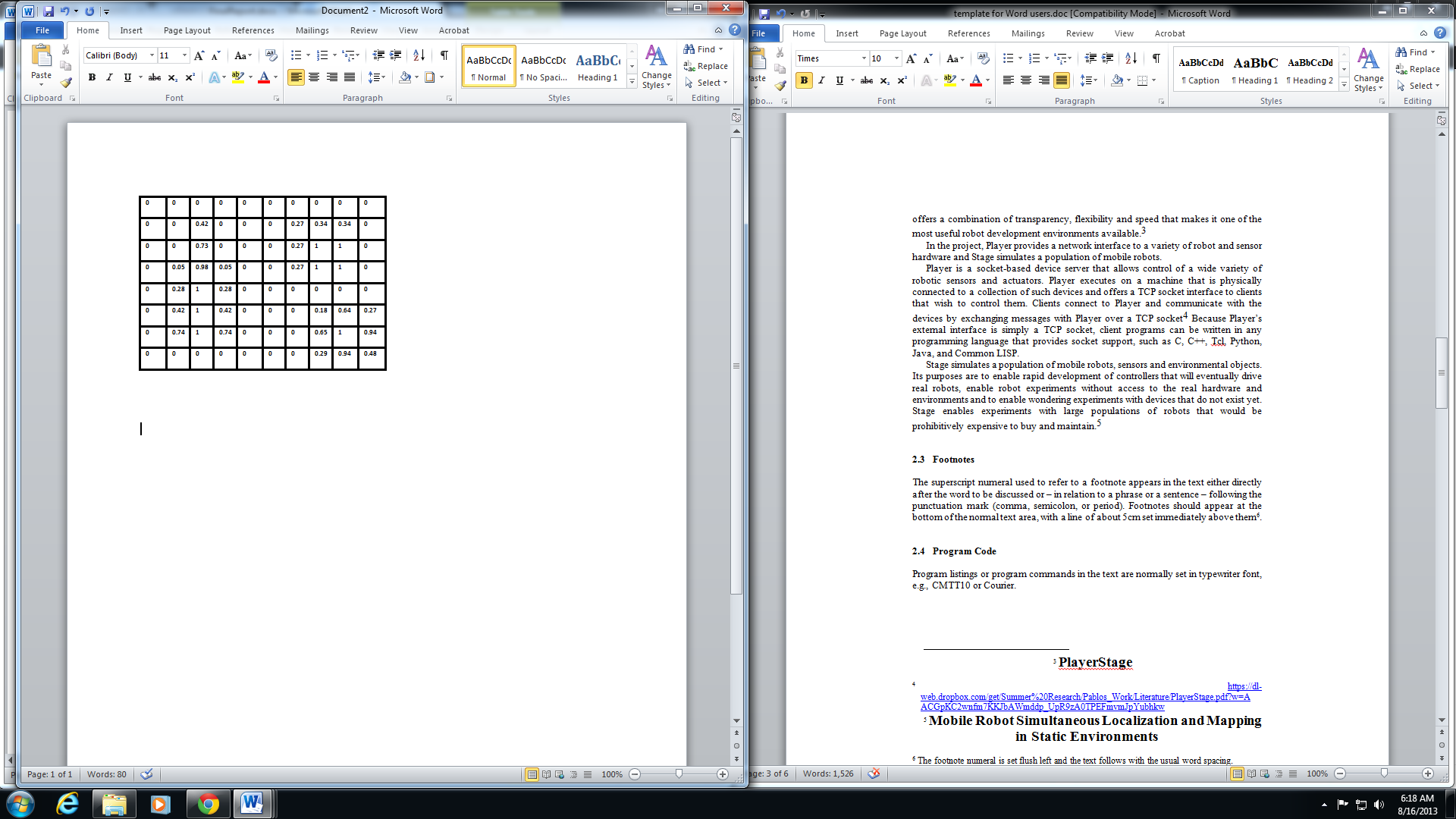
Fig. 2. An E-Puck robot with numerous components labeled (from [7])

5 Occupancy Grid

A popular and useful way that robots can map a physical area is by using occupancy grids. An occupancy grid representation employs a multidimensional tessellation of space into cells, where each cell stores a probabilistic estimate of its state [10]. In this project a two-dimensional area was correlated to cells in a two-dimensional matrix, each cell being filled with a value ranging from 0 (meaning that cell was certain to be empty) to 1 (meaning that cell was certain to be filled).



(a)



(b)

Fig. 3. Simple environment overlaid with a grid Fig. 3(a) and the corresponding two-dimensional matrix representing the occupancy grid of the environment Fig. 3(b)

The top of Figure 3 shows a two-dimensional environment containing a triangle, a square, and a circle. The environment is segmented into a grid. This grid correlates to the image on the bottom of Figure 3, which shows a two-dimensional matrix filled with numerical values ranging from 0 to 1. Areas in the environment that are completely filled, such as the center of the circle, have a corresponding value in the matrix of 1, while areas in the environment that are empty, such as the upper area of the environment, have a corresponding value in the matrix of 0. Cells that are not completely filled have a value between 0 and 1.

These values represent the probability that the cell is occupied, so that a value of 0 shows there is a 0% probability that the cell is occupied, and a value of 1 shows there is a 100% probability that the cell is occupied.

6 Experimental Setup

The task of mapping an environment requires translating the odometer and proximity sensor readings of the robot into a map. The Numpy and Matplotlib graphical libraries helped to visualize the map.

Figure 4 shows a robot and an object in a global coordinate system. The robot’s current position (X\_Robot, Y\_Robot) is assumed to be known. The robot takes proximity sensor readings (Range, Bearing). The robot’s task is to determine the coordinates of the object (X\_Object, Y\_Object) based on the robot’s known data (X\_Robot, Y\_Robot, Range, Bearing).

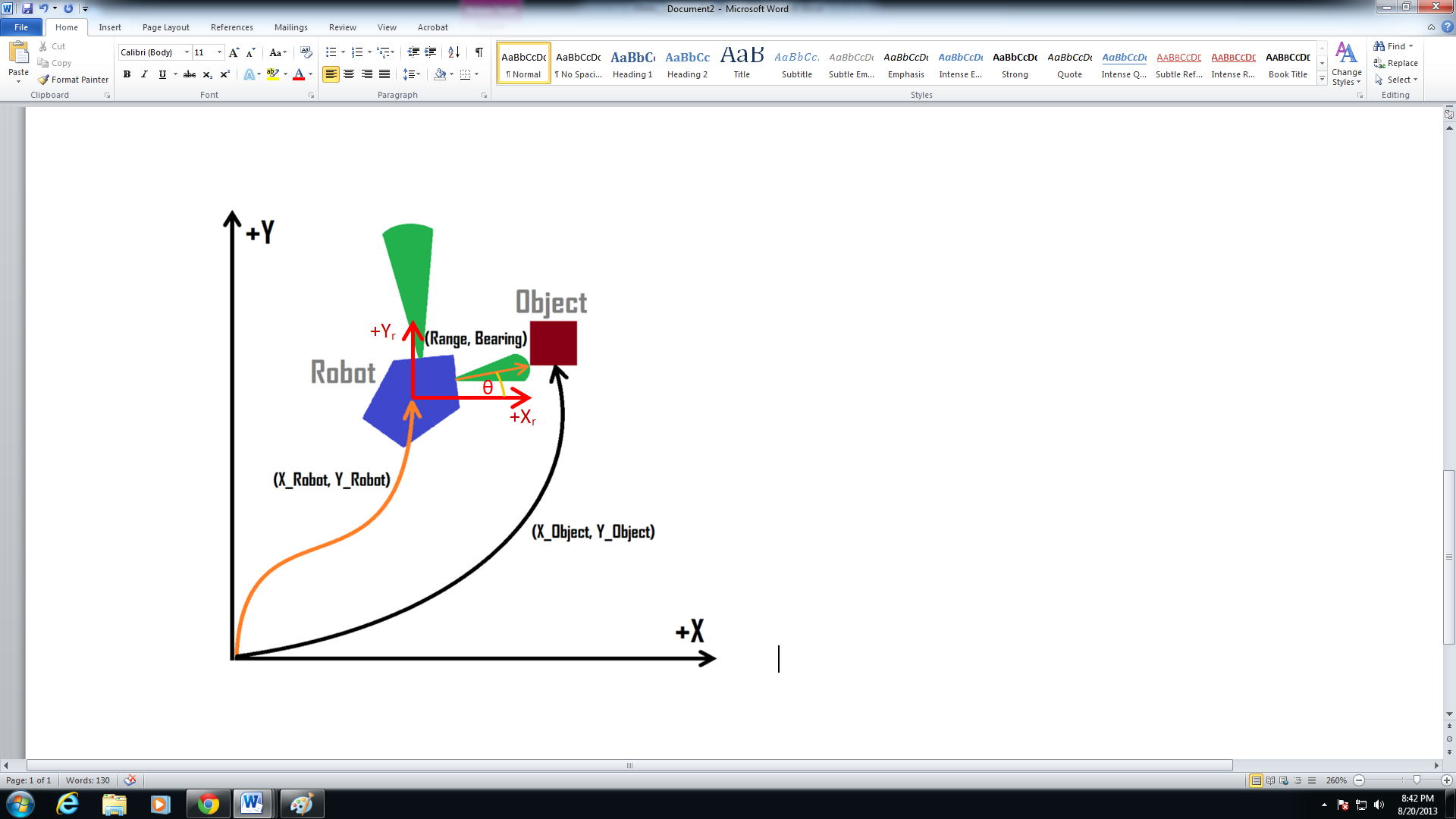


Fig. 4. A robot sensing the location of an object within a coordinate system based on its position and sensor readings.

Rigid body coordinate transformations are fully covered in [11]. The project simplified these transformations for a two-dimensional space as follows. Equation 1 shows the coordinates of the object with respect to the robot. Equation 2 shows the coordinates of the object with respect to the global coordinate system. Both equations have been condensed into matrix form.

|  |  |
| --- | --- |
|  | (1) |

|  |  |
| --- | --- |
|  | (2) |

In these equations, x\_objectRelative and y\_objectRelative describe the position of the object *relative to the robot*. Range refers to the numerical reading obtained from a proximity sensor on the robot. The value θ is the angle of the proximity sensor within the global coordinate system. X\_Objectand Y\_Objectdescribe the position of the object within the global coordinate system, and X\_Robotand Y\_Robotdescribe the position of the robot within the global coordinate system.

To determine the coordinates of an object within a coordinate system first the position of the object with respect to the robot is determined. For the object in Figure 4, this means using the numerical reading of the proximity sensor that registers the object and the value θ of the angle of this proximity sensor within the global coordinate system and applying these values to Equation 1. This then gives the coordinates of the object with respect to the robot.

The next step is to localize the object within the global coordinate system. This is a simple matter of adding the coordinates of the object relative to the robot (x\_objectRelative, y\_objectRelative) to the coordinates of the robot (X\_Robot, Y\_Robot).

For the project, the E-Puck robot could determine its position and bearing by means of the odometers in both of its wheels. The E-Puck robot’s eight infrared sensors gave the range of a detected object up to 6 cm away. The bearing of each proximity sensor reading was determined by two factors. First, the bearing of the E-Puck robot itself was determined, once again by the odometers in the wheels. Second, each of the eight sensors was positioned at a specific angle along the E-Puck robot. The angle on the E-Puck robot of a specific sensor was added as an offset to the bearing of the E-Puck robot. Figure 5 shows the angles of separation of each of the infrared sensors on the E-Puck.

|  |  |
| --- | --- |
|  |  |

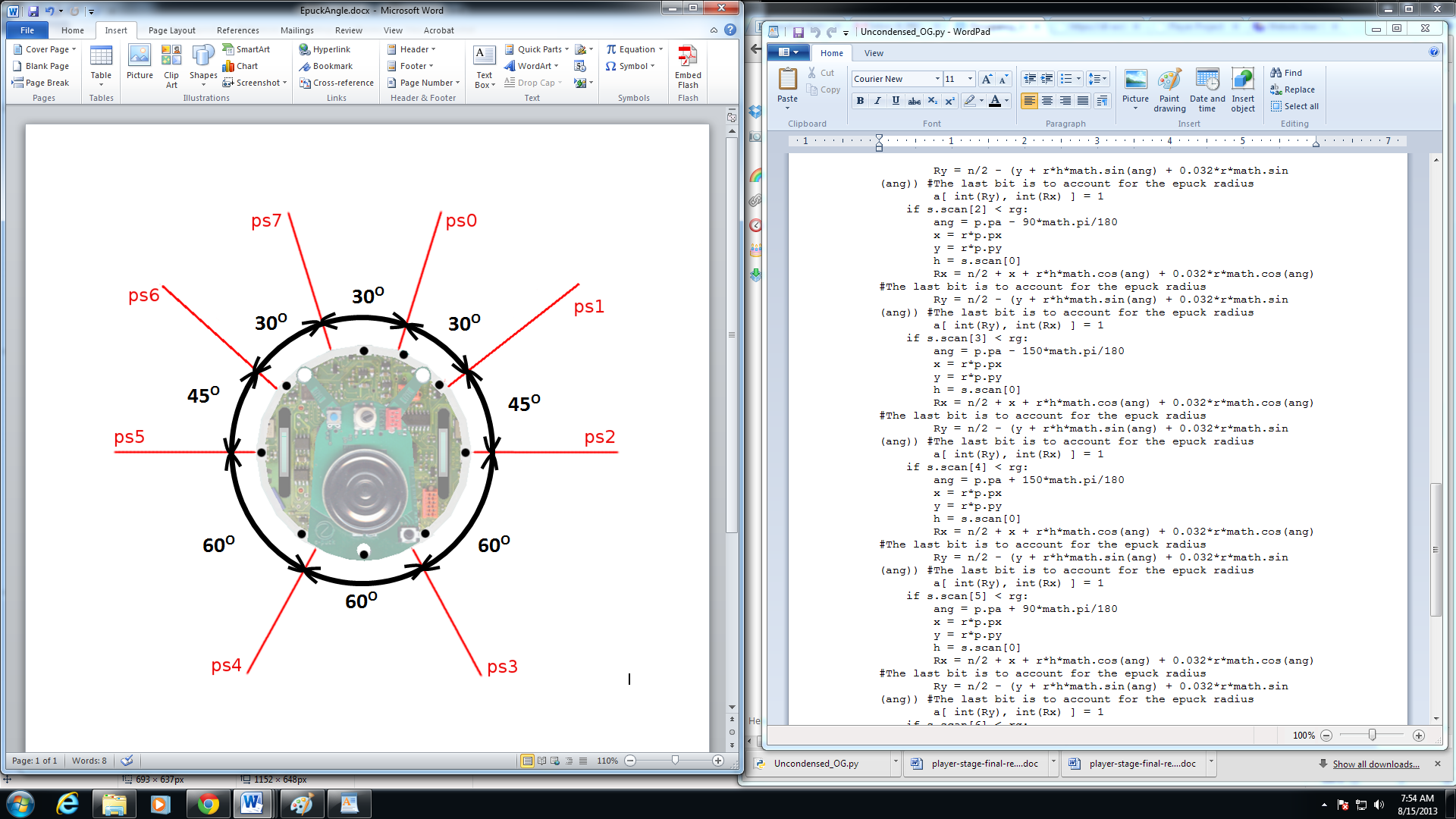


Fig. 5. Overhead view of an E-Puck robot with the angles between proximity sensors labeled (from [12])

In the lab, a physical environment 4 feet long by 4 feet wide (1.22m by 1.22m) was arranged. Figure 6 illustrates the physical environment. Likewise, a virtual environment was created in Stage that approximated the physical environment in both the dimensions specified and the arrangement of the shapes. Figure 7 illustrates the virtual environment.

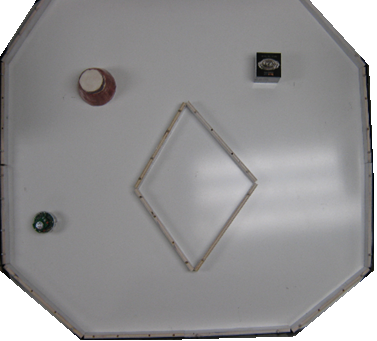


Fig. 6. Picture of the physical environment used for mapping. The E-Puck robot is seen in the bottom left.

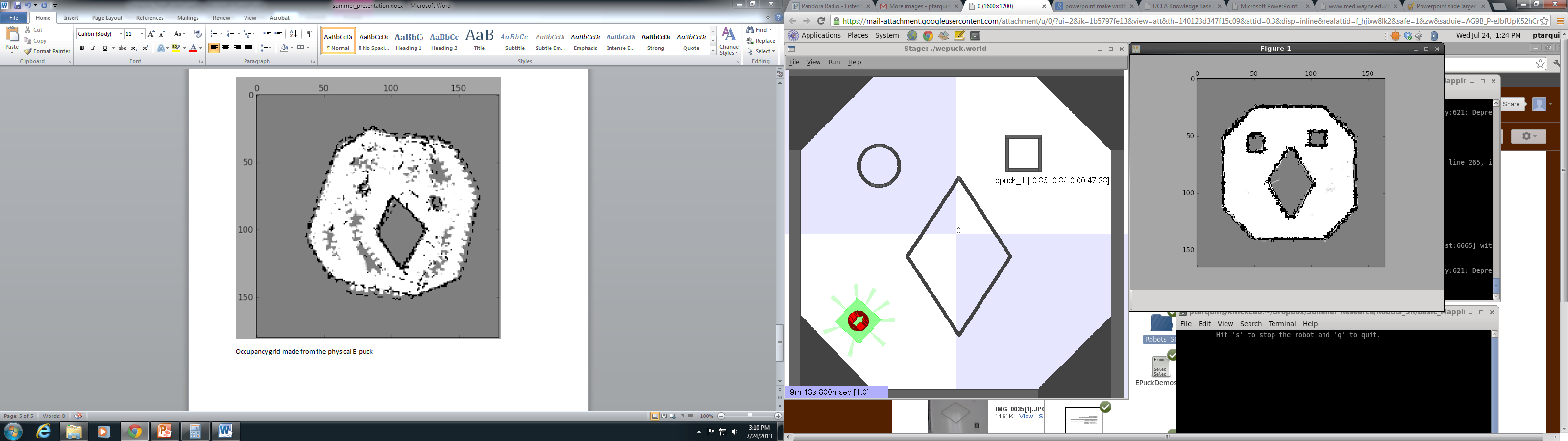


Fig. 7. Picture of the virtual environment used for mapping. The E-Puck robot is seen in the bottom left with eight proximity sensors “rays.”

7 Results of Mapping

The project used both a virtual and a physical E-Puck robot to create maps of the environments. In the virtual environment, the E-Puck robot had perfect localization conditions. For the physical environment, the E-Puck computed its coordinates from the odometer readings. These coordinate readings were subject to error, meaning the localization conditions were inaccurate.

7.1 Map for the Virtual Environment

The best maps were obtained within the simulated Stage environment. Figure 8 shows the map of the virtual Stage environment. This map closely replicates the virtual environment seen in Figure 7. During simulation, the location of the robot is exact. That is, Stage allows the simulated robot to have perfect odometer readings and eliminates the error caused due to the slippage of the wheels. The majority of the error can be seen around curved surfaces or sharp edges. This is due mainly to three factors. First, there are some inaccuracies in the readings of the range of the proximity sensors. Second, the size of the grid may not be large enough to give a good resolution of the map. Third, the wide angles of the proximity sensors make it hard to accurately depict small objects.

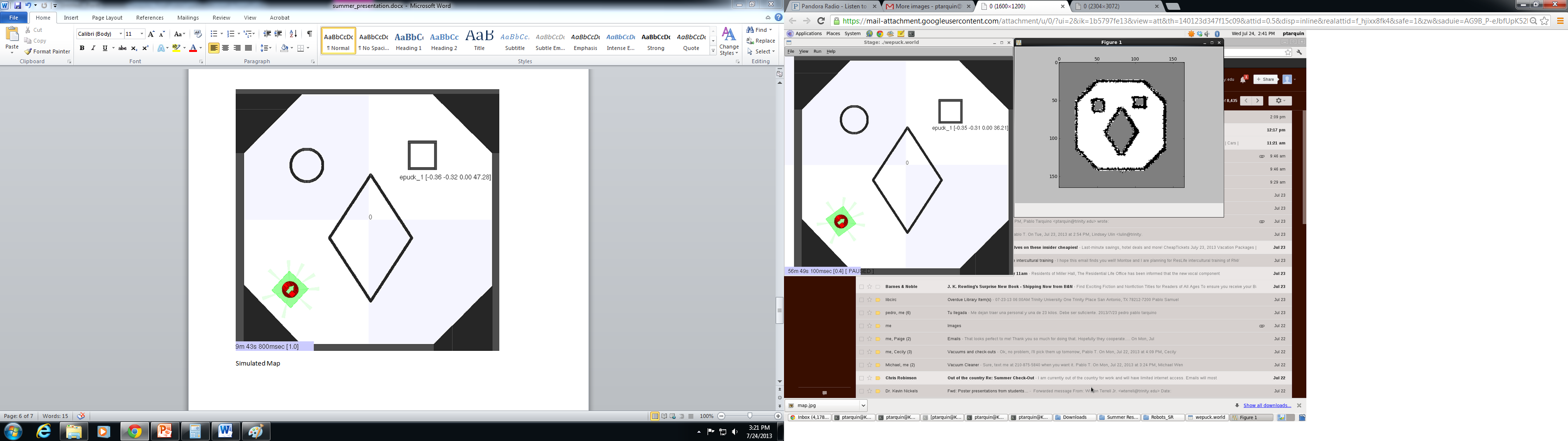


Fig. 8: Map of the virtual environment

7.2 Map for the Physical Environment

The results of the real E-Puck were less accurate than those of the virtual E-Puck. Figure 9 shows the map of the physical environment. As can be seen, this is an inaccurate representation of the environment shown in Figure 6. This inaccuracy is caused by error due mainly to well-known odometer drift [13]. The inaccuracies are greatly noticeable along surfaces that required the robot to turn sharply, causing wheel slippage, but they are less noticeable along flat surfaces. In this manner, surfaces that required numerous turns for the E-Puck robot to fully map them are greatly distorted, as is the case with the square and the circle found within the environment. Flat surfaces, such as the diamond or the borders of the environment, were mapped as fairly straight (albeit skewed) lines.

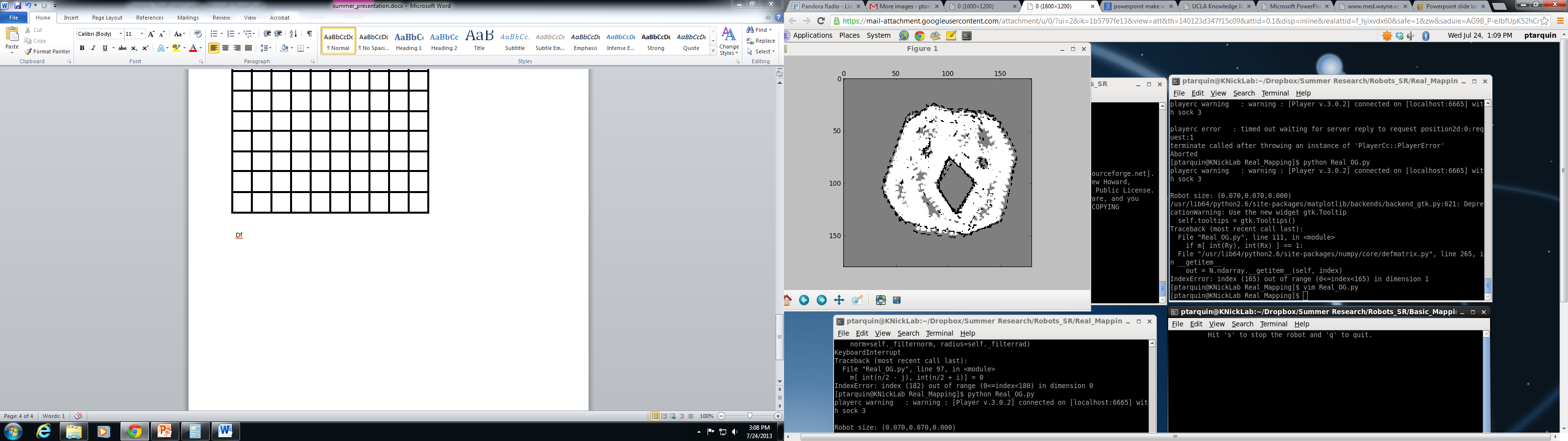


Fig. 9. Map of the physical environment

7.3 Accumulated Error for the Physical Environment

The E-Puck robot could be manually controlled, or it could be allowed to “wander” around the environment. Regardless of whether the robot was manually controlled or allowed to wander, the virtual runs produced maps that consistently looked like the map seen in Figure 8. The physical runs, however, required strict manual control. The most efficient routes had to be selected so that the E-Puck traveled along straight lines as much as possible. This is far from ideal, especially considering that the goal of SLAM is to allow a robot to make maps without any *a priori* knowledge.

To illustrate the detrimental effects of the localization error, the physical E-Puck robot was allowed to wander for some time; the results can be seen in Figure 10.

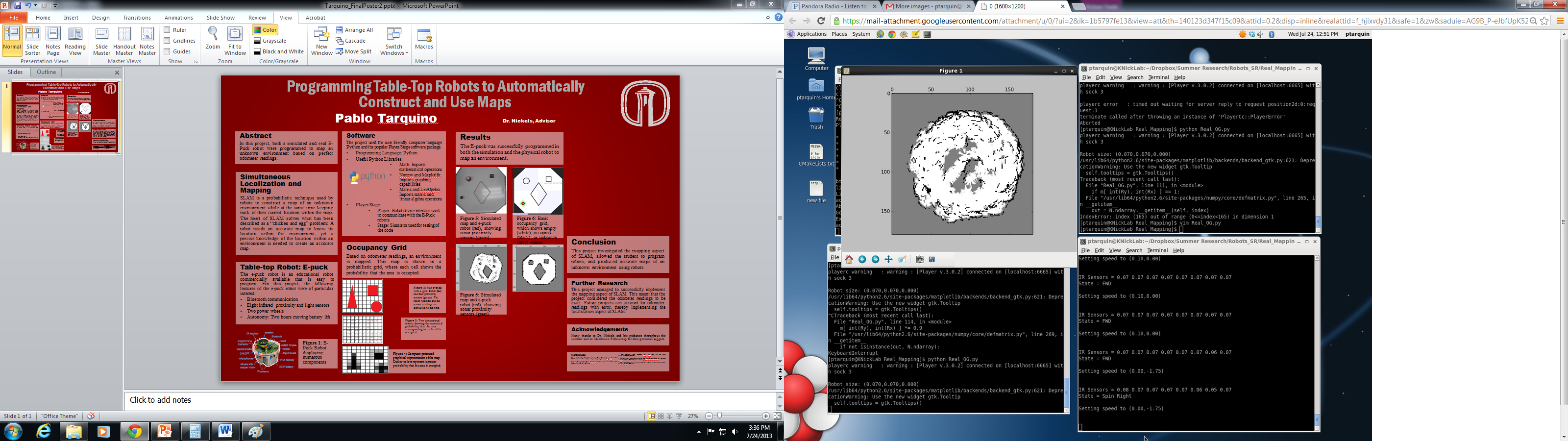


Fig. 10. Map of the physical environment when the robot was allowed to wander

8 Conclusions

The goal of this paper was to explore the mapping aspect of Simultaneous Localization and Mapping. An overview of SLAM and the theory used throughout the project was provided. This project ultimately produced very accurate maps when localization was exact, yet it also showed how inaccurate maps can result if localization is not exact. This illustrates the well-known effect of odometer drift on mapping, motivating further the SLAM problem.

This project was an excellent first step into SLAM research. It acquainted the student with the Player/Stage software and it allowed for hands-on programming of a robot.

Future research can focus on implementing SLAM techniques in order to integrate the localization aspect that was not covered throughout this project. This would allow a robot to accurately map an environment even with localization error, such as the physical scenario presented in this project.

References

1. Vaughan R., Howard A., Gerkey B.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003), pages 317-323, Coimbra, Portugal

2. Vaughan, R.: Massively Multiple Robot Simulations in Stage. In: Swarm Intelligence 2(2-4), pages 189-208, Springer (2008)

3. Bailey, T., Durrant-Whyte, H.: Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. IEEE Robotics and Automation Magazine, Vol. 2 (2006)

4. Bennet, S., Nieto-Wire, C., Peche, J., Timotheu, M., Vasili-Acevedo, D.: Mobile Robot Simultaneous Localization and Mapping in Static Environments. Technical Report, The City College of New York of the City University of New York

5. Zelle, J.: Python Programming: An Introduction to Computer Science. Franklin, Beedle, Oregon (2004)

6. Official Python Programming Language Website, http://www.python.org/

7. E-Puck Brochure, http://www.cyberbotics.com/e-puck/e-puck.pdf

8. Player Driver for E-Puck Robots, https://code.google.com/p/epuck-player-driver/

9. Basic E-Puck Robot Demos, http://www.gctronic.com/doc/index.php/E-Puck#Examples

10. Elfes, A.: Using occupancy grids for mobile robot perception and navigation. In: IEEE Computer, pages 46-57 (1989)

11. Wolovich, W.: Robotics: Basic Analysis and Design. Holt, Rinehart and Winston, New York (1987)

12. Webots User Guide, http://www.cyberbotics.com/dvd/common/doc/webots/guide/section8.1.html

13. Siegwart, R., Nourbakhsh, I.: Introduction to Autonomous Mobile Robots. MIT, Massachusetts (2004)

14. Mondada, F., et al.: The e-puck, a robot designed for education in engineering. In: Proceedings of the 9th conference on autonomous robot systems and competitions, Vol. 1. No. 1 (2009)

15. Thrun, S.: Robotic mapping: A survey. In: Exploring artificial intelligence in the new millennium, pages 1-35 (2002)