E-Puck Grid Ground Feedback Control and A Star Navigation

Ronald Picard
EECE 5257 Control Systems1
Vanderbilt University
Nashville, TN, USA
ronald.s.picard@vanderbilt.edu

A common problem in autonomous route planning and control is to implement efficient motor control, sensor feedback, and route planning algorithms to allow a robot to successfully navigate and traverse a line grid maze from a start point to a different destination point in an efficient manner. We present an E-Puck controller that utilized ground sensor feedback to perform a variety of functions; detect nodes (cross-sections), detect lines just past nodes, provide Braitenberg line tracking functionality, and allow for left and right turns on a line grid. In addition, we present an A Star algorithm that is utilized to efficiently solve for a route from a start node to an end node in a line grid maze. Finally, we present tuned parameters for node detection, line just passed node detection, line following, turning logic, and ground sensor threshold parameters resulting in a stable system.

Keywords— Human Detection, Neural Network, Classification, Security System, Raspberry Pi, Arduino, Artificial Intelligence, Automation

I. Introduction

a. Autonomous route planning and control is a complex field. The need to find algorithms for efficient motor control, sensor feedback, and route planning is prevalent. We present an approach to allow an e-puck robot to successfully navigate and traverse a line grid maze from a start point to a different destination point in an efficient manner.

II. Basics

A. *E-Puck 1.0*

The E-Puck robot is a small, lightweight, ground-based differential wheel robot that is equipped with 3 Infrared (IR) ground sensors, 8 IR distance sensors, wheel encoders, a front facing camera, and Bluetooth 2.0. [4] See figure 1 for a picture of e-puck 1.0

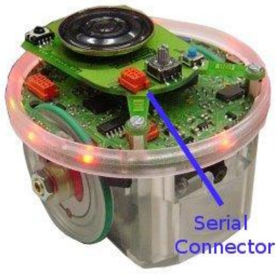


Figure 1: E-Puck 1.0 [4]

B. *Webots Software*

Webots is a simulation software, developed by Cyberbotics Ltd, used for the development simulation and control of virtual robots that performing tasks in a virtual environment. [1] Refer to Figure 2 for a screen shot of the Webots Software.

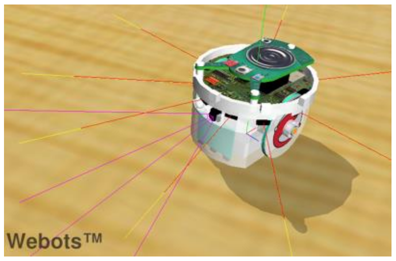


Figure 2: Webots Software [1]

C. *Differencial Wheel Robots*

Differential wheel robots are two-wheel robots that are holonomic (meaning able to rotate without moving forward). The E-Puck is a differential wheel robot that is able to maneuver forward backward and turn in a holonomic and non-holonomic fashion.

D. *Ground Sensor Feedback*

When tracking lines on a line grid three IR sensor are needed; a left sensor, center sensor, and a right sensor. These three sensors help to disambiguate where the line is at any given point in time. The E-Puck is equipped with a ground sensor extension pack. The sensor extension back comes with the IR sensors that are aligned horizontally to the front of the robot. Refer to Figure 3 for a picture of the ground IR sensors.

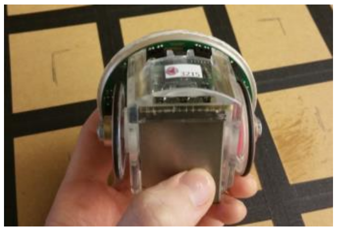


Figure 3: Ground Sensor Extension Pack

E. *A Star Algorithm*

The A Star algorithm is an algorithm that uses a cost heuristic function to rapidly determine a reasonable route to a destination given multiple paths. In the case of a line grid maze the this allows a route from one node (cross-section) to another node to be found and traversed in an efficient manner.

III. Objectives

A. E-Puck Feedback Control Objectives

The proposed approach was to write a C code controller that allowed an E-Puck robot to navigate through a series of node-line-node traversals on a line grid (e.g. 12x12). An indexed list the objectives for the E-Puck controller are listed below:

1. Create a control algorithm for the E-Puck that utilizes IR ground sensor feedback of the to follow a straight-line path.
2. Create a control algorithm for the E-Puck that utilized ground sensor feedback to detect when the robot has reached a node.
3. Create a control algorithm for the E-Puck that utilized ground sensor feedback to for turning right, proceeding forward, or turning left at a node.
4. Create an algorithm that utilizes the previous objectives to navigate a supplied series of line-to-node traversals on a line grid (e.g. 12x12).

B. A Star Algorithm Objectives

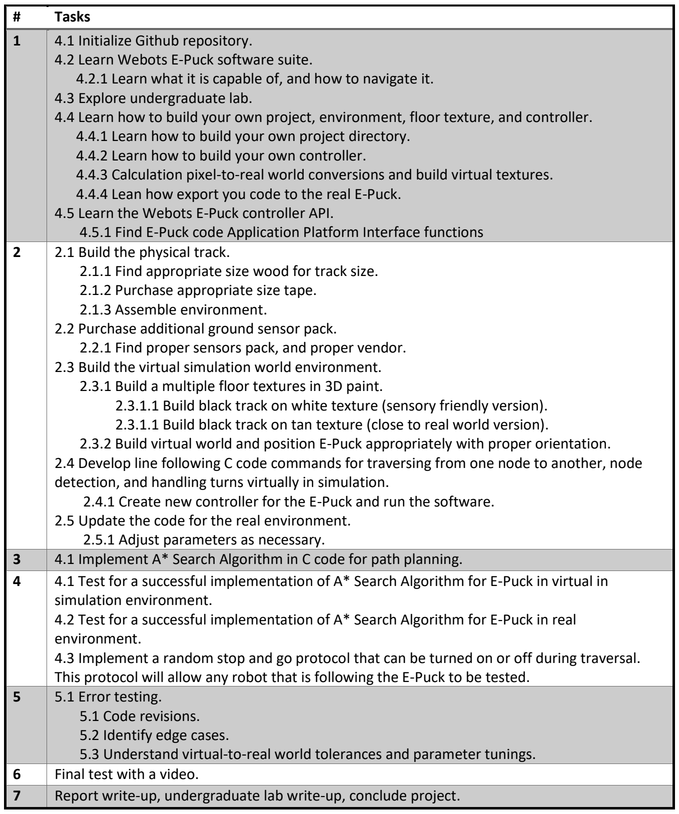
The proposed approach was to write an A Star algorithm in C code that allowed an E-Puck robot to navigate line grid maze utilizing the commands provided by Objectives Part A. An indexed list of objectives for the A Star algorithm are listed below.

1. Create an A Star Search Algorithm to solve a grid maze and return a set of direction from the start node to the end node.
2. Create an control flow logic to convert the A Star directions into robot commands developed in Objective A.4.

IV. Proposed Approach

A. Task List

Refer to Figure 4 for an indexed list of tasks for the proposed approach.



#	Tasks
1	4.1 Initialize Github repository. 4.2 Learn Webots E-Puck software suite. 4.2.1 Learn what it is capable of, and how to navigate it. 4.3 Explore undergraduate lab. 4.4 Learn how to build your own project, environment, floor texture, and controller. 4.4.1 Learn how to build your own project directory. 4.4.2 Learn how to build your own controller. 4.4.3 Calculation pixel-to-real world conversions and build virtual textures. 4.4.4 Lean how export you code to the real E-Puck. 4.5 Learn the Webots E-Puck controller API. 4.5.1 Find E-Puck code Application Platform Interface functions
2	2.1 Build the physical track. 2.1.1 Find appropriate size wood for track size. 2.1.2 Purchase appropriate size tape. 2.1.3 Assemble environment. 2.2 Purchase additional ground sensor pack. 2.2.1 Find proper sensors pack, and proper vendor. 2.3 Build the virtual simulation world environment. 2.3.1 Build a multiple floor textures in 3D paint. 2.3.1.1 Build black track on white texture (sensory friendly version). 2.3.1.1 Build black track on tan texture (close to real world version). 2.3.2 Build virtual world and position E-Puck appropriately with proper orientation. 2.4 Develop line following C code commands for traversing from one node to another, node detection, and handling turns virtually in simulation. 2.4.1 Create new controller for the E-Puck and run the software. 2.5 Update the code for the real environment. 2.5.1 Adjust parameters as necessary.
3	4.1 Implement A* Search Algorithm in C code for path planning.
4	4.1 Test for a successful implementation of A* Search Algorithm for E-Puck in virtual in simulation environment. 4.2 Test for a successful implementation of A* Search Algorithm for E-Puck in real environment. 4.3 Implement a random stop and go protocol that can be turned on or off during traversal. This protocol will allow any robot that is following the E-Puck to be tested.
5	5.1 Error testing. 5.1 Code revisions. 5.2 Identify edge cases. 5.3 Understand virtual-to-real world tolerances and parameter tunings.
6	Final test with a video.
7	Report write-up, undergraduate lab write-up, conclude project.

Figure 4: Proposed Approach Take List

B. Gantt Chart

The proposed schedule of tasks for the project was captures in a Gantt chart. Refer to Figure 5 for an a Gannt Chart of the project schedule.

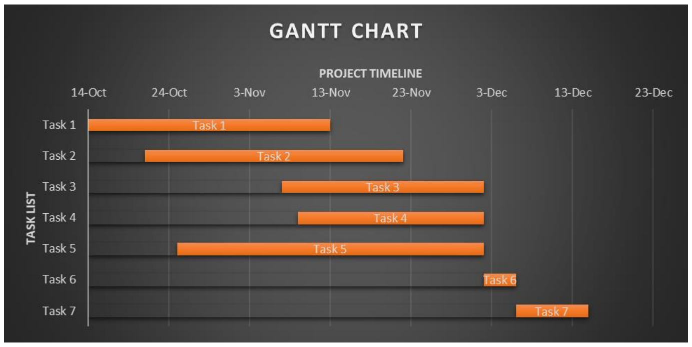


Figure 5: Gantt Chart

V. Description of COntrol System

A. Simulation Environement

The simulation environment was designing to mirror the real environment. A 12 by 12 line grid texture was created utilizing Paint3D. It was designed using pixel calculations to convert the real environment into a mirroring virtual image that could be imported into the Webots software. Refer to Figure 6 for the designed virtual texture.

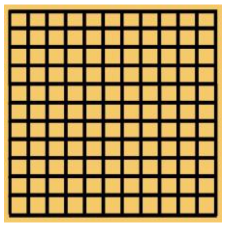


Figure 6: Virtual Texture

The Webots simulation environment comes with a virtual E-Puck module. Importing the Virtual texture into this environment and placing the E-Puck at the appropriate position and orientation provides a complete simulation environment in which the controller logic can be tested. Refer to Figure 7 for the virtual simulation environment. As illustrated the virtual simulation environment functions as an integrated development environment (IDE) that allows for the development of a visual simulation environment (shown in the middle of the image in Figure 7), as well as a C code controller for the E-Puck (shown on the right of the image in Figure 7). The C code can be compiler and run with the simulation environment. On the left of the image shown in Figure 7, the Tab labeled simulation may be switched to a Bluetooth COM port, to instead run the controller on the real E-Puck.

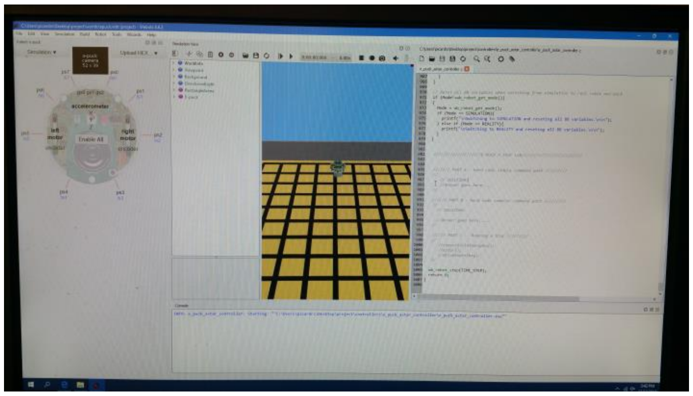


Figure 7: Webots Virtual Simulation Environment

B. *Physical Enviroment*

The physical environment is a mirror of the virtual environment. It consists of a large wooden board (approximately 4 ft by 4 ft) with a smooth surface. Double strips of 3/8 inch black painters tape, was used to develop 6/8 inch lines for the grid. Refer to Figure 8 for a picture of the physical environment.

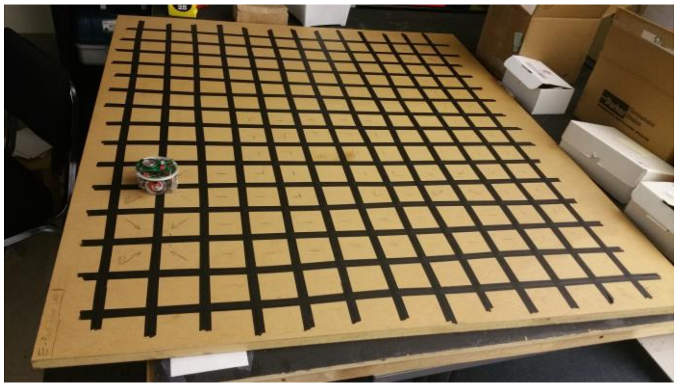


Figure 8: Physical Environment

The Physical environment was give extra tape on the edges of the nodes, so that the robot can detect a line just past a node on the edge of the board and perform a turn appropriately.

C. *Line Following Logic*

The line following logic is a simple Braitenberg algorithm. It takes the difference between the left IR ground sensor value and the right IR ground sensor value, multiplies the difference by a gain factor (between 0 and 1), then subtracts the values form a standard speed fixed-speed, and then assigns the speed it to a specific wheel. It then adds the difference value to the fixed-speed and assigns that value it to the other wheel. This ensures that if the robot begins to veer off in one direction, it will always correct itself to the other direction. The end behavior results in a sweeping motion of the robot. The Braitenberg wheel speed line tracking algorithm is bulleted below.

- LeftWheelSpeed = Standard_Speed-Gain*(LeftSensorValue-RightSensorValue)
- RightWheelSpeed = Standard_Speed+Gain*(LeftSensorValue-RightSensorValue

D. *Node Detection Logic*

The node detection logic is based off threshold values for the IR sensors readings. The values differ between the simulated board the and physical board. Because the tape on the board is black, and black tends to absorb light, the IR sensor reads a lower value for the black tape, than for the brown board. For each of the three IR ground sensors, if a sensor reads a value below the line threshold, then it determines that it is over a line. If a sensor reads a value above the board threshold, then it determines that it is over the board. In general, it is best to overestimate being on the board, and underestimate being on the line. Because the lines are only 6/8-inch thick, only two sensor values will ever be on the line at one time unless the robot has reached a node. Therefore, if three sensors values detect a line, the robot determines that it is at a node. When the robot first detects a node, it checks a few sensor consecutive sensor readings just to be sure that it has reached a node. This helps deconflict sensor readings as it drives onto a node. Once the robot has determined that it is on a node, it will drive straight until it detects the line passed the node. In order for this to work, it sets a flag that waits to be flipped until it detects that it has passed the node.

E. Line Just Passed Node Detection Logic

It is import for the robot to know when it has just passed a node and returned onto a line. When the all three sensor values no longer simultaneously detect a node, the robot determines that it is back on a line, flips the flag, and reinitiates the Braitenberg line tracking algorithm.

F. Turning Logic

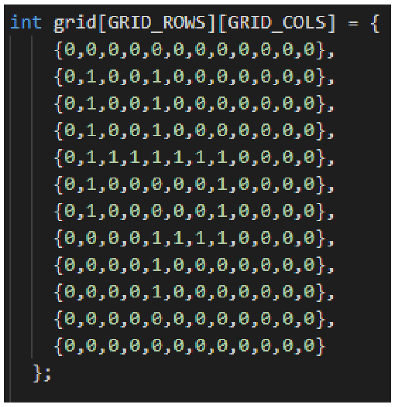
If the robot needs to make a turn, it waits until it is just passed a node, then it re-initiatiates the Braitenberg line tracking algorithm for a brief moment to move the robot away from the node and deconflict sensors readings during a turn. It then locks itself into a turning mode for a period of time while it turns, and its wheel speeds are set to fixed turning constants depending on the direction. The turn is nearly holonomic but not completely. While the robot is locked for turning it will ignore all sensor readings. This helps to deconflict the line it started on from the line it will end on. After a period of time (in the middle of the turn) the robot will unlock itself and respond to sensor values. At this point, as soon as the ground sensors values detect a new line it, the robot re-initiates the Braitenberg line tracking algorithm.

G. High-Level Command Logic

In order for the user to avoid dealing with the low-level complex logic of line tracking, node detection, and turns, a wrapper function was created to accept high level the commands. The wrapper function is named drive, and requires two arguments values to be passed into it; the number of nodes to traverse, and direction to traverse them in. The set of possible directions are left, right and straight; and the number of nodes up to the user. This allows a user to type a command such as drive(5, left), and the E-Puck will go to the next node, perform a left turn, and drive 5 nodes from there.

H. A Star Algorithm Logic

The A Star algorithm is implemented using a form a push pop stack que. A multi-dimensional array at the top of the file allows the user to visually specify a maze of nodes, where 1's represent paths that the robot may drive, and 0's represent walls. Refer to Figure 9 for an illustration.



```
int grid[GRID_ROWS][GRID_COLS] = {
    {0,0,0,0,0,0,0,0,0,0,0,0},
    {0,1,0,0,1,0,0,0,0,0,0,0},
    {0,1,0,0,1,0,0,0,0,0,0,0},
    {0,1,0,0,1,0,0,0,0,0,0,0},
    {0,1,1,1,1,1,1,1,0,0,0,0},
    {0,1,0,0,0,0,0,1,0,0,0,0},
    {0,1,0,0,0,0,0,1,0,0,0,0},
    {0,0,0,0,1,1,1,1,0,0,0,0},
    {0,0,0,0,1,0,0,0,0,0,0,0},
    {0,0,0,0,1,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0}
  };
```

Figure 9: Virtual Grid Maze Design

This multi-dimensional array is then parsed into a set of arrays that function as a graph, connecting possible directions that the robot may travel at any given node. This information is then fed to A Star algorithm which implements a push pop stack que with a heuristic cost function in order to find a solution to the maze in a reasonable amount of time and calculate the cost. The Heuristic for this A Star algorithm is provided below.

- | NeighborNodeX - GoalNodeX | + | NeighborNodeY – GoalNodeY |

Once the A Star algorithm has found path to the goal node it ceases execution.

I. Extracting A Star Solution Recusively

The A Star algorithm provides the first reference key to the first direction solution in the que upon completion. Each direction in the queue contains the previous key of the previous direction. These reference keys are used with a recursive algorithm to extract the path from the goal node back to the start node of the E-Puck from the que. After extraction, the array containing the solution directions is backwards. Therefore, an algorithm is used to reverse the array so that it can more easily be fed into the high-level command logic of the controller.

J. Converting the A Star Global Directions into E-Puck Local Orientation Commands

The A Star algorithm uses a global fixed reference frame for the grid (up, down, left, right). These must be converted to the relative orientation of the E-puck as high-level commands as it traverses the grid. Control flow logic was written to handle this issue when driving the robot. The function loops through the A Star solution direction array and corrects for the relative orientation of the E-Puck along the way. It does so by searching forward in the solution array for relative repeat commands and adjusts the direction and number of nodes to traverse for each line accordingly.

K. Adding Random Pauses

A random pause flag was designed that, when enabled, causes the robot to make up to one random driving pause (for a random amount of time) during each straight line of nodes traversed. This is desirable for when other robots that track the E-Puck need to be tested if they maintain a proper distance.

VI. System Performance

A. *Physical E-Puck Statistics*

When the E-Puck runs in the physical environment the results are not deterministic like they are in simulation. Therefore, the E-Puck was tested to analyze performance. The Left, Right, and Straight line logic methods were each tested 25 times to analyze their failure rate. The test of the straight line was for the E-Puck to follow a straight line accoss the entire physical board. The test for the left and right turns were for a successful turn. A successful turn is where the robot leaves the line just passed a node and re-initiates the Braitenburg line tracking alorgithm on the left or right lines, respectively. The results of this analysis are indexed below.

1. Straight Line Following Tests:
    a. Successful Rounds: 25
    b. Failed Rounds: 0
    c. Failure Rate: 0%
    d. Success Rate: 100%
2. Left Turn Tests:
    a. Successful Rounds: 25
    b. Failed Rounds: 0
    c. Failure Rate: 0%
    d. Success Rate: 100%
3. Right Turn Tests
    a. Successful Rounds 23
    b. Failed Rounds 2:
    c. Failure Rate: 4%
    d. Success Rate: 96%.

The successful results from the tests perform granted confidence to the E-Pucks ability to traverse the grid.

B. *IR Ground Sensor Thresholds Simulation Environement*

The ground sensors IR threashold values for the Simulation enviroment differ from the physical environement and when either one was used for the other, this resuled in significant line tracking errors. The IR threashold values for the simulation envirnment were found to be most stable with the following.

1. Board Detected > 700
2. Line Detected < 600

C. *IR Ground Sensor Thresholds Physical Environement*

The IR threashold values for the physical envirnment were found to be most stable with the following.

1. Board Detected > 550
2. Line Detected < 400

D. *A Star Complex Traversal Stability*

Many errors were debugged once the A Star algorithm began to drive the E-Puck on complicate routes. The Braitenberg algorithm gain factor was scaled down to 0.3 to increase stability of the line tracking by decreasing the sweeping motion range. A minor pause delay was initiated right after the robot returns onto a line off a node; and after the brief pause the Braitenberg line tracking algorithm initiates for a brief moment to move the robot away from the node to deconflict sensors readings during a turn. These two steps became an effective technique for stabilizing turns. With these additions the E-Puck rarely failes when traversing complex routes.

VII. Conclusions

We have presented an E-Puck Controller which utilizes ground sensor feedback to traverse a 12 by 12 line grid. Controller logic was design which utilizes the IR ground sensors feedback to detect nodes (cross-sections), detect lines just past nodes, provide Braitenberg line tracking functionality, and allow for left and right turns. In addition, A Star algorithm logic was implemented to efficiently solve for a route from a start node to an end node in a line grid maze. This route was then passed to the e-puck to allow the epuck to traverse the route. Error testing was perform to tune the line following, turning logic, and ground sensor threshold parameters resulting in a stable system.

Acknowledgments

We would like to acknowledge K. Kawamura for his project idea, technical support, and class lectures on feedback control systems. We would like to acknowledge T. Darrah for his support, advice, and guidance regarding the A Star algorithm implementation.

VIII. References

[1] https://cyberbotics.com/#webots , 'Why Webots', 2018. [Online]. Available: https://cyberbotics.com/#webots [Access: 13-Dec-2018]

[2] http://www.e-puck.org/ , 'e-puck education robot', 2018. [Online]. Available: http://www.e-puck.org/ [Access: 13-Dec-2018]

[3] http://www.e-puck.org/ , 'e-puck education robot', 2018. [Online]. Available: http://www.e-puck.org/ [Access: 13-Dec-2018]

[4] http://www.gctronic.com/, 'e-puck education robot', 2018. [Online]. Available: http://www.gctronic.com/doc/index.php/E-Puck [Access: 13-Dec-2018]

[5] https://www.mathworks.com/products/simulink.html. [Accessed: 11-Dec- 2018].