CMSC 479/679: Class Project

For this class, the project will consist of building, programming, and ultimately demonstrating tasks being performed a GoPiGo 3 platform. The platform consists of two actuated wheels, a body, a laser-based distance sensor, and a Raspberry Pi that serves as the "brain."

Overview

In order to make sure everyone is on track, there will be a series of milestone deliverables, which will be a mix of individual and group assignments. Details about each milestone will be updated when the previous one is due, and this document will be updated over time.

0. Form groups, fill out group survey (tiny.cc/robotics-survey)

1. Build and proof of life
    - Write up *individual* responses to a survey about building
    - Demo robot working using included software
        - Remote control
        - Programmatic control

2. Install Linux-based OS and start Python control
    - Install Raspbian
        - Optional: Install ROS (see details on this milestone)
    - Programmatic readouts from all sensors
    - Programmatic control of all servos
    - Wall-following (initial)

3. Demonstrate maze navigation
    - Wall-following in a simple maze
    - Demonstrate initial maze-navigation capability using, e.g., random walk
        - Localization and mapping!
    - Submit overview of methodology and intended approach (*679 only*)

4. Final code submission due
    - Improve maze-following (better sensor use? Better AI? Different sensors? This will be where you can really make it your own!)
    - Initial writeup of design decisions and methodology (*679 only*)

5. Final writeup due (templates will be provided)
    - Overview of progress
    - Video of behavior

6. In-class competition/demos!

Notes

A note on group work: **it is really important that everyone be involved in every stage of the process**. If your group is having trouble (either you're having trouble working with your group, or there's someone in your group who is having trouble meeting and participating), *please* let me know as soon as possible, so we can fix it.

Milestone 0

We will start building robots on March ~~15th~~ 27th, in class. Before then, you should have finalized your group and hopefully begun meeting. When your group is finalized, please fill out tiny.cc/robotics-survey. We will take attendance on that day.

1. Build your robot: www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/1-assemble-gopigo3
2. Attach the distance sensor: www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/4-attach-the-camera-and-distance-sensor-to-the-raspberry-pi-robot

Milestone 1

Our first milestone will be construction of the robot and "proof of life" (that is, a demo of it working and doing robot things). We will be working from this robot,[1] and using DexterOS to begin. You may wish to start reading: assembly instructions at *https://www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/1-assemble-gopigo3.*

Step 1: Connect and Rename

3. Connect to your robot: studio.dexterindustries.com/cwists/preview/1218x
4. Immediately rename it to something unique (consider including your team name or an unusual last name) by following the instructions in this video: www.youtube.com/watch?v=uao88f_n5Kk

   → *Note: If we are all working in the same physical space, we will have to do this one at a time. Since we're doing this step in class, I will coordinate this. If you do this outside of class, make sure you communicate with other teams working nearby.*

5. Reboot!
6. ~~**Email** the professor and TA the MAC (hardware) address of your robot's wireless card:~~

   ~~a. SSH into your robot, using its new name, e.g.: `ssh dex-prof.local`~~

   ~~b. Run `ifconfig` on the command line and find the hardware address on the wlan0 device — it will be a sequence of six two-digit hex codes separated by spaces, like: `0D:43:A1:[etc]`~~

   → *You will do this step in Milestone 2; it only works in Raspbian.*

Step 2: Control and Proof of Life

7. Attach to the DexterOS interface (remembering to use your new wifi name), as in step **3** above.
8. Controlling the robot:

   c. **Manually:** Go to "Drive" and drive the robot around, using the arrow keys and space bar.

   d. **Programmatically:** Go to "Code in Python" and perform some of the projects listed. Make sure the process of simple robot control in Python is comfortable.

   → *Note: because we are doing this in class, you do not need to submit any kind of video; just show me that you have it working.*

9. Do your writeup of the build and control process at: tiny.cc/build-writeup.

[1] https://www.dexterindustries.com/shop/gopigo-advanced-starter-kit/

Milestone 2

Our second milestone will be installing Raspbian on your robot, and beginning the process of controlling the robot using a more complex infrastructure. (More details after midterm!)

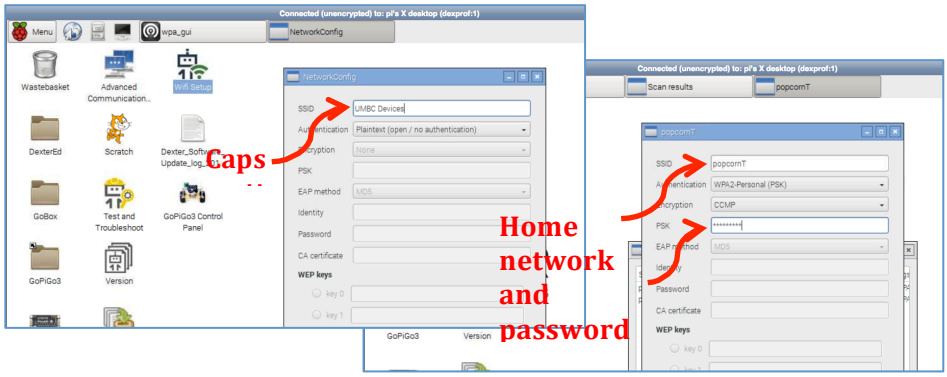
Step 1: Install Raspbian

1. *Carefully* retrieve your microSD card from your robot. This can be ticklish—you may have to do some disassembly to avoid cracking the card; go slowly and patiently.
2. Follow the instructions for installing Raspbian on the card:
   www.dexterindustries.com/howto/install-raspbian-for-robots-image-on-an-sd-card

   → *Note: The most recent .rar file in SourceForge seems to be broken; I had to go back to the previous version. Please keep us informed if you encounter similar difficulties.*
3. Reinstall the SD card in the robot.

Step 2: Connect to the Robot

4. You will need to first connect over Ethernet to configure its wireless setup:
   www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/2-connect-to-the-gopigo-3/Raspbian-For-Robots-Operating-System
   a. Plug an Ethernet cable into your computer or a wired or wireless switch or router (*not* one of UMBC's, please). I will have one in class, and can do this if we get this far.
   b. Start the VM image (in a browser on the computer).
   c. Follow the instructions in the link above to "SET UP WIFI." You will want your robot to know how to connect to two networks: `UMBC Devices` (for when you are on campus), and your home network. See the screenshots below for what this should look like.



   d. On the robot, edit the file `/etc/wpa_supplicant/wpa_supplicant.conf` . Under the line **`ssid="UMBC Devices"`**, add the line: **`scan_ssid=1`** ; save and quit
5. Change the name and password of your robot to something unique:
   www.dexterindustries.com/howto/change-the-hostname-of-your-pi
6. **Email** the professor and TA the MAC (hardware) address of your robot's wireless card:
   a. SSH into your robot, using its new name, and run `ifconfig` on the command line and find the hardware address on the wlan0 device—it will be a sequence of six two-digit hex codes separated by spaces, like: `0D:43:A1:[etc]`

So far, you have been following relatively tightly defined steps for the project. From here on out, you will have a lot of flexibility in how you want to proceed. This means you should think about the effects of your design decisions and start thinking about the high-level architecture you want to implement. Any reasonable implementation can get full credit; however, some choices will be easier and/or more successful than others!

Step 3: Infrastructure Design

7. Start considering your software architecture. You're going to want sensors, motor control, decision-making, and any other functions (like mapping) in a single, coherent software infrastructure. How will these pieces fit together? Discuss the options and write a short (~half page) description of your approach that addresses *at least* the following points:

   a. What is the overall architecture of your system?

   b. What are you using for development? Does anyone have an IDE? Does everyone?

   c. Are you using ROS to pass messages? If not, how are components communicating?

   d. How are team members sharing code and making sure everyone can do development—github, bitbucket, or something else? (Do use a version control system of some kind.)

8. Install ROS on your robot (*optional*).

   a. Using ROS for the remainder of the project is optional. However, if you want really good performance, it is recommended, because ROS has a number of tools that will be useful. As an incentive, note that you are allowed to use any standard ROS tools or in any of the experimental gopigo nodes, and (with permission) other nodes you find. This includes servo controls and ways of getting messages from the sensors, but also odometry, mapping, localization, several kinds of SLAM, and visualization in rviz.

   b. Instructions for installing ROS under Jessie: wiki.ros.org/ROSberryPi/Installing%20ROS%20Indigo%20on%20Raspberry%20Pi

Step 4: Control the Robot

Background information:
The following web pages will give you an overview of programming the GoPiGo3 and has a number of useful links, including to the GoPiGo3's github page and examples of how to use the encoders and servos:

- www.dexterindustries.com/GoPiGo/get-started-with-the-gopigo3-raspberry-pi-robot/3-program-your-raspberry-pi-robot/python-programming-language
- gopigo3.readthedocs.io/en/master

You may import `easygopigo3`. You may use code found in any of the tutorials or code provided by Dexter, *with proper attribution.* This means when you use their code, you must include source clearly in the comments of each file *and* at the top of each function.

9. In the architecture you defined above, write functions to control the servos on your robot.

   a. Turn the distance sensor some number of degrees (specified by an argument) right/left.

   b. Turn wheel 1 or 2 forward or backward independently (this can be several functions, or a single function that takes arguments).

   c. Control the wheels together to move the robot forward/backward.

d. Control the wheels together to turn the robot 90° right/left.

e. Control the wheels together to turn some number of degrees (specified by an argument) right/left.

f. Turn the wheels in order to move the robot a specified distance forward or back (in cm).

Step 5: Read from Sensors

10. Write functions to:

a. Get a single reading from the distance sensor.

b. Get a continuous stream of readings from the distance sensor.

c. Read the encoders' position, in degrees. (See github.com/DexterInd/GoPiGo3/blob/master/Software/Python/easygopigo3.py)

d. Print the encoders' positions in a continuous stream.

Step 6: Put it Together

11. In order to demonstrate all of the above, write a short routine that performs the following steps (without any human intervention) and take a video. The console should be printing the output of the distance sensor 4 Hz.

a. Approach a wall head-on until the robot is 10cm away.

b. Rotate 90° right.

c. Rotate the servo to point to the wall.

d. Proceed forward along the wall another 40cm.

e. Note: during step d, the robot does not need to perform any course correction to remain an exact distance from the wall (that will be the next milestone). However, if the robot is in danger of crashing into the wall, it should stop; something should be monitoring the distance sensor.

Turnins for Milestone 2

1. All code written for this milestone, in a form that can be put on our robot and tested.

2. A writeup with the following sections:
- The overall design and architecture decisions you made in Step 3, including code management and version control. Consider including a link to your repo.
- A short description of how to install your code and run the demo you wrote for Step 11. (Unzip it somewhere on the robot? Run a ROS node installer? ..?)
- Any code sources that you used and what code you incorporated or referenced.

3. A short video of your robot performing its demo.

Milestone 3

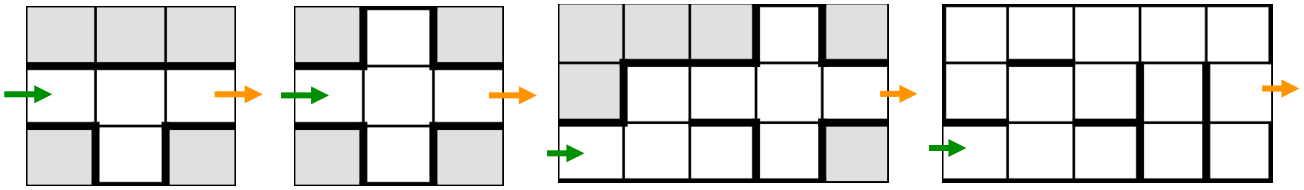
Our third milestone has two major parts: Decide on a strategy for the remainder of the project (and write it up); and demonstrate your robot doing basic uninformed maze-following, plus initial work on pursuing your strategy.

Step 0: Build a Maze

Your code will be much easier to debug from now on if you actually have a maze to work in. The easiest thing to do will be to build a maze (usually from cardboard on a cardboard base). Sample mazes are shown below.

The mazes we are working with will:

- Be laid out on a grid of squares 11 inches/28 centimeters wide, which will give a well-centered robot room to turn.
- Always have one or more legal paths.
- Have walls higher than the default mounting of the distance sensor.
- Contain dead ends and wrong paths, as well as one or more correct solutions.
- Have loops.



map 1 *map 2* *map 3* *map 4*

These sample mazes—arranged from least to most complex—cover a lot of cases. You may construct one or more of these, or design your own. You shouldn't build all of these, and for this milestone, I do not recommend building something very complicated.

Step 1: Implement Random Exploration

The simplest form of maze solving is *random exploration*. The robot maintains no state about the maze (such as a map); instead, every time a navigation choice is encountered, it makes a random decision about what path to take. This approach is guaranteed to solve the maze given infinite time; for simple mazes it works well in practice, for very complex mazes it is intractably slow.

Your first step is to implement such a random solver.

1. Identify decision points.
    a. In order to perform a random exploration, the robot must be aware when there are opening(s) requiring it to make a decision. This means turning the robot or sweeping the servo to be aware of walls and openings.
    b. Making random navigation decisions and successfully executing those choices.

2. At each decision point, randomly choose a direction.
    a. The robot does not need to consider reversing course (that is, turning 180°) at every decision point. (At dead ends that is the only option.)

b. At every decision point, you must consider all the possible directions the robot could go.
    i. At T-junctions, there are 2 choices; and four-ways, there are three choices. At a turn or a dead end there is only one choice.

→ *Note: To be clear, for this milestone, the navigation **must** be random.*

3. Implementation details:
    a. Your code should have some form of readout that clearly shows when the robot thinks a decision point is reached, what the choices are, and what the robot chose.
    b. You will lose points for bumping into the walls.
    c. Although the robot should go fast enough to complete the mazes above in a reasonable amount of time (~5 minutes), you will not gain points for going faster. Don't concentrate on speed; concentrate on sensor data and movement precision.
    d. We will put the robot down in entry, and when it exits the maze, pick it up. The robot does not need to find the entry or recognize that it has exited.

Step 2: Design Decisions

4. ***Mapping and localization.*** In order to pursue a more complex strategy, you need to decide whether you are handling the following problems: *mapping* and *localization*. Mapping means building and maintaining a map of the space as the robot traverses it; localization means maintaining a belief about where the robot is at all times. You must consider whether you wish to do mapping and/or localization, and if so, how to implement it.
    a. What will your knowledge representation look like? (Will you maintain a map laid out on a grid? Will you keep track of the geometric relationship of intersections?)
    e. Will you write an implementation, or use some existing implementation of mapping, localization, or SLAM?

5. ***Strategy.*** Based on whether you have mapping and localization information, exactly how are you handling the maze solving? What's the most efficient approach?
    a. With a physical agent:
        i. Actual movement is expensive and slow
        ii. Every intersection you go past is a chance for sensor failure
        iii. The farther you go the worse your localization gets

    An "efficient" strategy, therefore, minimizes these things, notcomputation.
    b. Remember that we are working with very simple mazes. A lot of the complexity you will encounter will have to do with successfully parsing sensor data, performing localization, and managing movement without bumping into walls. Don't go overboard coming up with something amazingly clever.
    c. Don't try to do this from scratch! Do some literature searching. Don't forget to take notes so you can cite anything useful you find.

Turnins for Milestone 3

1. All code written for this milestone, in a form that can be put on our robot and tested, with a short, useful README ***text*** file describing how to run it.
2. A PDF writeup with the following sections:

- A short description of how to install your code and run the demo you wrote for Step 11. (Unzip it somewhere on the robot? Run a ROS node installer? ..?)
- Any code sources that you used and what code you incorporated or referenced.

3. A video of your robot navigating a simple maze. (The first maze shown above is fine, as is anything more complex.) Make sure the video includes the output showing the decision points encountered and the decisions made. This should be less than five minutes and, if your maze is complex, doesn't need to show the entire solving process.
4. A PDF writeup of the strategy you have chosen, including your intentions for localization and mapping and a how you plan to implement them. All further writeups will be extensions of this document, so consider structure carefully.
    - *479 only:* This should be a ¾- to 2-page, single-spaced document, including any graphics, written in complete sentences, with meaningful section headers.
    - *679 only:* From here we will be working in conference paper format. Download either the LaTeX or Word template from www.roboticsconference.org/information/authorinfo (look under "Paper Formatting") and use it for your writeup. Add section headings for (at least) Introduction, Approach, Experimental Evaluation, and Conclusion. The design decisions you make in this milestone should go into Approach.

Milestone 4

At this stage, you should have a robot that is successfully navigating a maze, making random choices at intersections, and maneuvering smoothly without bumping into walls. The next step is to implement a more informed approach: the strategy you described in Milestone 3. From this point on, project milestones become much less specific.

Tasks for this milestone are: (1) improve your robot in clever, effective ways; and (2) write up the decisions you've made.

Goals:

It can be helpful to consider how we will grade the final work. Note that your robot will not be graded *primarily* on its ability to quickly execute a maze, but you do need to accomplish this task to get a good score. We will also grade based on the clarity and elegance of your code, the correctness of your approach, and how well your strategy captures appropriate robotics concepts in solving the problem. We will consider four main areas, which also appear in the writeup: moving, sensing, strategy, and core robotics concepts.

- ***Successful movement:***
    - Does the robot bump into anything?
    - Does it stay centered in the corridors, and if not, why not—is it by design or by accident?
    - Does it traverse the maze *relatively* quickly? Speed is low priority, but it does have to finish.
- ***Successful sensing:***
    - Does the robot use its sensors appropriately?
    - What happens if you get unexpected / unexpectedly noisy sensor inputs?
- ***Maze strategy:***
    - Is the strategy you have chosen appropriate for the complexity of the problem?
    - Are there (legal) cases your robot can't handle?

- How well does your approach handle unexpected circumstances (for example, really bad sensor data?

- ***Robotics concepts:***
  - Are you using (or not using) localization, mapping, and SLAM appropriately?
  - Are you using appropriate belief and/or map representations?
  - Are you using a suitable mix of behavior-based and location-aware decision-making?
  - Are your solutions informed (by this class) and elegant, or hacked together?

Step 1: Implement Improved Maze Navigation

By now you should have a pretty good idea how you're going to implement a solution to the overall maze problem. You should be making final decisions about your strategy, including how (and whether) to apply the tools we've learned over the course of the semester. Your strategy at this point should *not* be random.

1. Implement an initial (working!) approach to the overall maze-solving strategy your group has chosen, including steps such as localization and mapping.
2. Demonstrate the working implementation in a test setup. This should be complex enough to fully test the strategy you've chosen.

Step 2: Write Up Work

Your previous writeup should have given you a good infrastructure; at this stage, you should be filling out the writeup with text about the details of the approach you took. Your paper should have:

3. At least an *outline* of a related work section, introduction, and conclusion. Related work should include correctly formatted citations for any code you incorporated or referenced, as well as any papers or sources.
4. An experimental evaluation section, containing an *outline* describing how you're testing your robot and what you're finding so far.
5. A *written* approach section, which should describe your overall approach. In addition to what your robot is doing, at least touch on most of the following points:
   - ***Strategy:***
     - Is your approach deterministic?
     - How efficient is your chosen solving algorithm?
     - Have you altered your robot? If so, how and why?
   - ***Robotics:***
     - Are you localizing? If so, using what approach? What belief representation are you using? If not, why not, and what are you doing instead?
     - Are you mapping? What map representation are you using? If not, how are you solving efficiently?
     - Are you performing SLAM? How?
     - Is your approach partly or fully behavior-based? In what way(s)?
     - What else from this class are you incorporating?
   - ***Movement:***
     - How are you calculating what moves to make (e.g., choosing angles for turns)?
     - Have you calibrated out or otherwise adjusted for any errors? How?
     - How are you handling actuator noise?
   - ***Sensing:***
     - What sensor data are you using?

- Have you calibrated out or otherwise adjusted for any errors? How?
- How are you handling sensor uncertainty?

Turnins for Milestone 4

- All code written for this milestone, in a form that can be put on our robot and tested, with a short, useful README ***text*** file describing how to run it.
- A video of your robot navigating a more complex maze. This should be less than five minutes and doesn't need to show the entire solving process. If there is informative console or audio output, include it.
- A PDF writeup with the sections described above.
  - *479 only:* By now this should be a 3- to 4-page, single-spaced document, including any graphics, written in complete sentences (or a bulleted outline where noted), with meaningful section headers.
  - *679 only:* Continue working in conference paper format; make sure your responses to the points above are integrated in appropriate sections. All references throughout should be cited appropriately and appear in a bibliography.

Milestone 5

Please note: this has been split into two deliverables. The majority of it is due on the evening of May 14th; however, the "Experimental Evaluation" section of the writeup can be incomplete at this point (but does not have to be). The completed version is due the evening of the 17th. See Step 2, below.

This milestone is the final version! You will turn in your final code base for your robot, a *descriptive* video, and a complete, research-paper style writeup. Your goals are the same as Milestone 4: Have a robot that elegantly handles maze navigation in a way that is robust to error, makes full use of sensor information, has a good strategy for robotic maze solving, and incorporates what you've learned in class. You should not change your core approach.

Although this turnin gives you an additional week and a half to do big hunting and minor improvements, your basic strategy should be nailed down and implemented at this point. The biggest block of your time should be spent on the writeup.

Step 1: Finish Robot Navigation

The code you turn in at this stage is what will be running on your robot at the final maze-running demo. There are no specific additional guidelines; at this point you know what you want your robot to do. However, here is some ***advice***:

- **Concentrate on basics.** Because this is the final turnin, before experimenting, make sure the baseline is solid, because your actual strategy depends on the core working. Can your robot:
  - Move forward in a straight line? Backwards?
  - Move forward a fixed amount (say, 28cm)? Backwards?
  - Turn 90° either direction?
  - Reliably tell how far it is from side walls?
  - From the end of the corridor ahead?
  - Center itself between two walls?
  - Turn a corner? Without bumping anything?

- Notice an opening in a side wall?
- Navigate through an opening?

- When that's working reliably, **practice working through simple mazes**.
  - Can your robot reliably—say, 5 times in a row—navigate through the four example mazes given, or some similar variant?
  - If you simulate noise (for example, nudging the robot to one side while it's moving, or briefly passing a hand in front of the sensor), what happens? (No, I won't do that. You're just testing out noisy cases.)
- **Don't worry too much** about weird cases.
  - Most of the mazes will be pretty simple. You're getting graded on your strategy, use of everything we've learned in class, and general robotic robustness.
  - You aren't getting graded on your ability to figure out what curveballs we might throw at you; the more complex mazes are mostly for fun.
  - It's not a good use of your time to worry about it. It *is* a good use of your time to be able to handle basic mazes.
- **Practice good software engineering**.
  - Even though (actually, especially when) it's close to a deadline, it is a really good idea to break what you need to do down into functions, write unit tests, and pay attention to architecture.
  - Meet with your group consistently and when you have agreed to. Use email. Make sure everyone knows who is doing what pieces. This ***will*** affect your grade.

Step 2: Write Up Work

The final iteration of your paper is a project report. This will probably be on the order of 8-10 pages; it shouldn't be shorter than 6 or longer than 12 unless you have a very specific reason.

- The majority of what you write will probably be in the Approach and Evaluation sections. Very broadly, the first contains the description of strategies, plans, architecture, and decisions you made, and the second contains the actual results of trying to make it work in the real world.
- Feel free to include pictures and figures such as architecture diagrams. In fact, please do—it generally makes your paper a lot more readable and informative. Always use meaningful captions.

Your paper should contain (at least) the following sections:

1. **Introduction:** Describe the project (very briefly), then a high level, conceptual overview of your approach and your experimental results (that is, how well it worked out). This is typically about a page.
2. **Related work:** This should contain a short (1–2 sentence) description of sources you accessed, including any libraries, github projects, our textbook, useful web pages, videos you watched, etc. Please note that this is not just a list—it is written text, and reading it should give me an idea what these sources are and what they contributed to this project. The actual links or sources should be cited in this section, and formatted as references in the bibliography. (See example below.) Don't forget to include libraries like easygopigo.
3. **Approach:** This is the section where you tell us all about everything interesting you did and all the design choices you made, as well as your thought processes and other considerations. This should build on all the points from Milestone 4. What did you end up doing? What decisions did you make and why? What other approaches did you consider, and why did you do them? (If you tried them and it didn't work in practice, that goes in the next section.)

4. **Experimental evaluation/Results:** A description of how you're testing the robot and what happened. Did you build a maze? (Include pictures!) What did you have to do to make it work? Did you calibrate the robot? How? If you elect to include the final demos in your writeup, how were the mazes set up?

   An explanation of what actually happened during testing. Did the robot navigate successfully? If not, what happened? Did you try something and have to switch approaches because it didn't work when you tested it?

   This is a good place to expound on the real-world difficulties and decisions you made. Did your robot consistently bump into walls? In practice, did your strategy perform poorly in a maze where all the openings were on the left? What went well?

   NOTE: *If you wish,* you may turn in an updated version of your paper with a results section that talks about how the robot did in the actual maze demos. This is intended to give you the opportunity to describe your results in the actual final "working" conditions, as well as in your own testbed.

5. **References:** These should be in a separate section at the end. Any standard citation format (APA, MLA, or IEEE) is fine; just be consistent. (679 students should use the template format.) For specific guidance, see: `http://tiny.cc/ieee-citations`

Example related work:

> For this project, we explored several different ways of implementing SLAM [1]. We initially intended to use FuzzySLAM [2], which is designed to handle mapping in a maze, but it turned out to be unsuitable because it made our robot grow actual fuzz, which interfered with the servos (see Figure 3). We also considered using an existing particle filter-based localization implementation [3], but decided to build our own implementation of mapping when we remembered the environment is just a bunch of squares.

Example bibliography:

> [1] Dissanayake, M. G., Newman, P., Clark, S., Durrant-Whyte, H. F., & Csorba, M. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3), 229-241, 2001.
>
> [2] Matuszek, C. and Smith, T. FuzzySLAM: Using Fuzzy Logic on Completely Inappropriate Problems. *Journal of Improbability Distributions in Robotics*, 2011.
>
> [3] Smith, J. J. J, Particle Filters on A Tiny Robot. GitHub repository, https://github.com/smithjjj/ros_gopigo3_particles. [Retrieved May 1, 2018].

Turnins for Milestone 5

- Your final, complete code, in a form that can be put on our robot and tested, with a short, useful README ***text*** file describing how to run it.
- A video of your robot navigating a more complex maze. This should be less than five minutes and doesn't need to show the entire solving process. If there is informative console or audio output, include it.
- A PDF writeup with the sections described above.