CSCI 3302: Final Project Submission Documentation

Vienna Wong | Email: viwo3022@colorado.edu
Kevina Wong | Email: kewo2098@colorado.edu
Sahand Setareh | Email: sahand.setareh@colorado.edu
Jaskrit Singh | Email: jaskrit.singh@colorado.edu
Group's main communication channel: https://discord.gg/qU2RyDc4
Demo Video: https://youtu.be/oCt879c837Y

<u>Abstract</u>

Our final project will use Natural Language Processing to take a voice command and have a group of ePuck robots perform a group action. Specifically, we are going to have the robots create shapes and figures given a voice command such as "Circle, Square, Triangle, Stick Figure, or Integral." Given these commands, the ePuck robots will create the shape as a group while avoiding collisions.

<u>Equipment</u>

- WeBots Robot Simulator
- Epuck (16 separate)

Note: We did not pursue an implementation using LIDAR on each ePuck that we initially anticipated

<u>Deliverables and Implementation Plan</u>

✔ - Indicates that it works & completed
✘ - Doesn't work/ Incomplete; Fallback Used

[✔] **Create World -** Lead: Kevina Deadline: 11/30/20

- <u>Implementation Plan:</u> We plan on making a world with not much in it so the robots can move around. We will also be having about 16 epucks.
- <u>Sub-Deliverables:</u>

  [✔] Create World
  [✔] Setup 16 ePuck Robots in initial positions with Lidar sensors mounted
  Implementation Description: Created a 2m x 2m world in WeBots and added 16 ePucks to the world.

[✔] **Program Robot Controller**

- <u>Implementation Plan:</u> For this implementation, we plan on splitting up each subdivision of the code for a different member of the team. These specific sub-deliverables are listed below.
- <u>Sub-Deliverables:</u>
  - Implement code for movement to various shapes - Lead: Jaskrit Deadline: 11/30/20

    [✔] Map shapes layout (defining position goals making up of each of the five shapes: Circle, Square, Triangle, Stick Figure, Integral) - Kevina

    Implementation Description: In WeBots, I moved the ePucks into a shape and recorded their translation values.

    [✔] Define map resolution that makes shapes look good (to make sure shapes look good) - Kevina

    Implementation Description: We found their coordinate frame difficult to work with, so we changed it to where the bottom left would be (0,0). To account for this change, we changed the translation values to x = x+1 and y = (-y) + 1.

    [✔] Decide which robot moves where and avoid conflicting goals - Kevina

    Implementation Description: Created a dictionary where keys are the initial pose as the key and a tuple of the target pose and a batch number. We sorted robots and targets in terms of their distance from the origin and mapped the robot closest to the origin to the target closest to the origin and the robot farthest from the origin to the target farthest from the origin. We also gave the robots a "batch number" according to their distance from the origin. We send the robots one at a time 10 seconds after each other according to the batch number. This ensures that we fill the robots in a good order that avoids all collisions.

[✔] Swarm communication setup - Jaskrit

Implementation Description: We made a server that communicates with all of the robots and the user input script. The user input script can post the target coordinates to the server. The server also allows each robot to post its coordinates and makes a cookie to keep track of the different robots. If targets have been posted and we have enough robots, the server maps each robot to a target. Robots can then make get requests for their targets and if the time for the robot to move has come and the mapping is done, they will receive a target pose.

[✘] Avoid obstacles and other e-pucks - Vienna

Implementation Description: We did not implement obstacle avoidance in the robot controller because the issue of avoiding e-pucks was resolved with another solution. The solution is explained in the "decide which robot moves where and avoid conflicting goals" sub-deliverable. Since e-pucks are mapped to a specific goal and sent in various batches, it avoids the collisions of robots completely.

[✔] Navigate to goals to create shapes - Vienna

Implementation Description: Once the e-puck reaches the state where it has received it's target pose and is ready to head to it's goal destination, heading to the target is done by creating a feedback controller. Initializing the robot's initial position and the target's position, the controller uses inverse kinematics that takes an e-pucks initial position and helps it navigate to its desired position.

Test: Each robot moving from starting position to target position

Test: Two robots on a collision course -> resolve collision

Test: E-Puck swarm create desired shape given shape coordinates

- Implement Natural Language Processing for parsing voice commands - Lead: Sahand Deadline: 11/30/20

[✘] Receive voice commands

Implementation Description: We encountered a few obstacles with implementing voice command capabilities for our robots. The first of which was inaccurate interpretations of shape names, as well as module import incompatibilities with outdated versions of python libraries pyaudio and portaudio. Instead, we opted for the fallback option of having a terminal-input shape provided by the user, with the same consequent process of parsing the command for the shape names and relaying them as a command to the robots. The user-requested shape relays information to the server that in turn sends information to the robots.

[✔] Parse commands for keywords

Implementation Description: Terminal input from the user is received to be compared to preset shapes in a shape dictionary. If this does not match a shape in the shape dictionary, the user is prompted for a new shape.

[✔] Parse word into shape

Implementation Description: Terminal input from the user is checked to be a shape with given target coordinates constituting the shape that is to be formed by the robots.

Test: Translate from spoken language to robot commands

Test: Accurate detection of keywords and phrases

- Implement code to convert shape to N coordinates (if time) - Lead: Jaskrit Deadline:12/5/20

  [✘] Convert drawn figure to N coordinates

  [✘] Define all corners/intersections

  [✘] Use extra points in places where shape could use more definition (ex. Center of square edges)

  Test: Convert given shapes to coordinates

  Test: Covert some other shapes that we draw in paint

<u>**Fallbacks and alternatives:**</u>

[✔] If dealing with collisions between ePucks becomes too difficult, we may make the robots move to their appropriate locations one after another.

Implementation Description: See "Decide which robot moves where and avoid conflicting goals " Subdeliverable Implementation Description.

[✔] If there is not enough time to accommodate the swarm shapes of stick figure and integral, then we resort to just doing the three basic shapes of circle, square, and triangle.

Implementation Description: It took us a long time to get our current implementation to work for the three shapes we defined, and with time constraints we decided to keep our implementation to circles, squares and triangles.

[✔] If the strategy to employ natural language processing to define our swarm shapes falls short of our expectations, we will resort to a user-type terminal input to define our shapes instead

Implementation Description: Because of difficulties that arose with the processing accuracy of our text-to-speech approach and module import incompatibilities with outdated versions of python libraries pyaudio and portaudio, we opted to use a user-type terminal input to define the shape that the robots were to make

<u>**Demo:**</u>

We will be demonstrating our complete robots using a video demonstration of the swarm understanding an example terminal command and successfully swarming to the appropriate shape. For example, if the user inputted “circle”, the robots will form a circle