Assignment 0

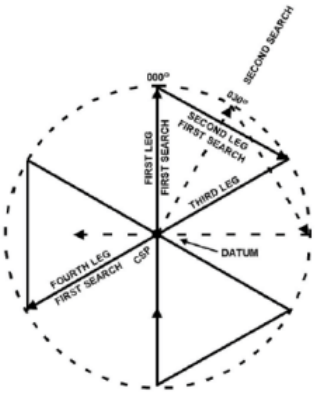
Introduction

The aim of the assignment is to setup the environment (guide to setup) and try out hands on the environment to practise and test the setup.

Turtle Graphics

Requirements

To encourage the setup of a working Python environment, create a pattern using turtle graphics. Python has a built-in turtle graphics module. To make things more interesting, our dimwittedturtle has lost his friend, and needs to execute a *Victor Sierra* search pattern, shown below



Code

How to Run the code

Open a new terminal and cd into `HW0 - Turtles/Turtle Graphics` directory. Run the Python file using:

```
python3 turtle_graphics.py
```

Code Basics

```
def main():
    search_spread = 50  # Radius of search
    active_search = False  # Whether it is an active search or simple pattern build
```

- `search_spread` variable is used to set the search area radius for the Vierra Pattern to build. This distance is in *pixels*.
- `active_search` is a boolean which determines whether to actively look for the lost turtle or just create a Search Pattern for visual display.

```
def perform_search(rescuer_turtle, lost_turtle, search_radius, active_search):
```

Move the turtle in Vierra Search Pattern. If the rescuer is in:

- active rescue state: it move 4 unit in forward direction and looks for the lost turtle.
  - If it fails to find it, it continues.
  - If it finds the lost turtle, it returns the current location of rescuer turtle and the error of position
- Not Rescue State: Creates Infinite Vierra Search Pattern

Args:

- `rescuer_turtle (turtle.Turtle())`: The Turtle Object Looking for the Lost Turtle
- `lost_turtle (turtle.Turtle())`: The Turtle Object that is lost
- `search_radius (int)`: Search Area Radius in px
- `active_search (bool)`: True → Active Search, False → Pattern Building

```
def victor_sierra_search(rescuer_turtle, lost_turtle, search_radius, active_search):
```

Initiate a Victor Sierra Search Pattern.

Args:

- `rescuer_turtle (turtle.Turtle())`: The Turtle Object Looking for the Lost Turtle
- `lost_turtle (turtle.Turtle())`: The Turtle Object that is lost
- `search_radius (int)`: Search Area Radius in px
- `active_search (bool)`: True → Active Search, False → Pattern Building

```
def place_lost_turtle(lost_turtle, lost_radius, active_search):
```

Randomly select the position to set the location of lost turtle. Turtle is hidden if it is not active search

Args:

- `lost_turtle (turtle.Turtle())`: The Turtle Object that is lost
- `search_radius (int)`: Search Area Radius in px
- `active_search (bool)`: True → Active Search, False → Pattern Building

Results

Active Search:
Click the Video below to see the simulation:

Inactive Search:
Click the Video below to see the simulation:

ROS Installation

We need to install ROS Noetic. The complete installation guide can be found here.

ROS Demonstration with TurtleSim

Open a new terminal and initialize roscore using:

```
roscore
```

Open a new terminal and run the following code to initate the turtlesim:

```
rosrun turtlesim turtlesim_node
```

Initiate the script to read the keyboard keys to control the turtle using:

```
rosrun turtlesim turtle_teleop_key
```

Use your arrow keys to move the robot around.

Get the list of all active topics in a new terminal using the command

```
rostopic list
```

Echo the `turtle1/pose1` topic using:

```
rostopic echo /turtle1/pose
```

The Terminal will update with the latest position and velocity values.

```
x: 3.077805519104004
y: 4.504674911499023
theta: -2.8480000495910645
linear_velocity: 0.0
angular_velocity: 0.0
```

Create an Obstacle Field

Subsequent assignments will rely on planning within two dimensional *grid world* type environments. Using any tools or programming languages, create a random obstacle field in a 128×128 grid, using tetrominoes as depicted in Figure.

"/HW0 - Turtles//Resources/Tetromino.png" is not created yet. Click to create.

Create a function, $f(\rho)$, to randomly distribute obstacles with varying coverage $\rho \in [0, 1]$. For example, a coverage rate of 10% would place approximately 400 obstacles in the field, occupying maybe 1600 cells out of 16384 total. Demonstrate your obstacle field implementation by submitting full source code, and three figures depicting 10%, 50%, and 70% obstacle coverage.

Code

Setup Dependencies

This graphical interface is build using Simple and Fast Multimedia Library (SFML) Library. It is a well designed and efficient library with required rudimentory functions needed to build simple graphical displays.

To install the library, paste the following code in terminal

```
sudo apt-get install libsfml-dev
```

Now, there is a huge and boring process needed to make sure that your file is linked with SFML Library. So I have made a *MakeFile* to make the task simple.

If you want to try out things for your own. Here is the link.

How to Run the code

Open a new terminal and cd into `HW0 - Turtles/Obstacle Field` directory. Run the file using:

```
make
```

Wait for the compilation to finish. The window will prompt out your input for the coverage percentage required `Enter The Area Coverage Percentage:`

Code Basics

In order to improve the speed of the code, a small trick has been applied. We have roughly calculated the required blocks that are needed to be placed.

This is a sample calculation show below:

$$h = 128$$
$$w = 128$$
$$h * w = 16384$$
$$\eta = 10\%$$

where η is the coverage percentage.

$$n = 1638$$

where n is the number of pixels to be covered.

$$n_{blocks} = \frac{1638}{4} = 409$$

We randomly place 409 block and do not tally or count the grid total as considering overall of blocks and placement, the total coverage still shall remain under 1638 pixels. After the minimum number of pixels are placed, we check the grid total and thus, calculate the total of the grid and then place new blocks after constantly checking the total. This way, for the first 409 turns, we don't check the total sum and are able to reduce this step and save some time.

```
#include <SFML/Graphics.hpp>
```

Header File to include SFML Library after installation.

```
#define GRID_WIDTH 128
#define GRID_HEIGHT 128
std::uint8_t grid_array[GRID_WIDTH][GRID_HEIGHT]; // Grid to store info
```

To set the `GRID_WIDTH` and `GRID_HEIGHT` in px length. This determines the size of bounding box.
`grid_array` stores the bit information of where the particular position on the grid holds a block or not.

```
void setup_grid(sf::RenderWindow *window)
```

Set the up the Graphical Window to display the Matrix by building the horizontal and vertical lines to define the matrix box. Also Set the Frame Rate to 30fps.

Takes the pointer to `window` as input.

```
std::array<uint8_t, 2> get_block_placement_position()
```

Generate a random position and return a (x,y) coordinate array.

```
uint8_t get_block_type()
```

Select a random block type out of the four available option.

"/HW0 - Turtles//Resources/Tetromino.png" is not created yet. Click to create.

Return number according to Block Type:

- `0` → Line
- `1` → Inverted L
- `2` → S
- `3` → T

```
void update_grid_window(uint8_t block_type, std::array<uint8_t, 2> location, sf::RenderWindow *window)
```

Set the block on grid window. Takes block type, location and window as input.

```
void update_grid_array(uint8_t block_type, std::array<uint8_t, 2> position)
```

Set the bit on the grid array. Takes block type, location and window as input.

```
std::uint64_t calculate_coverage()
```

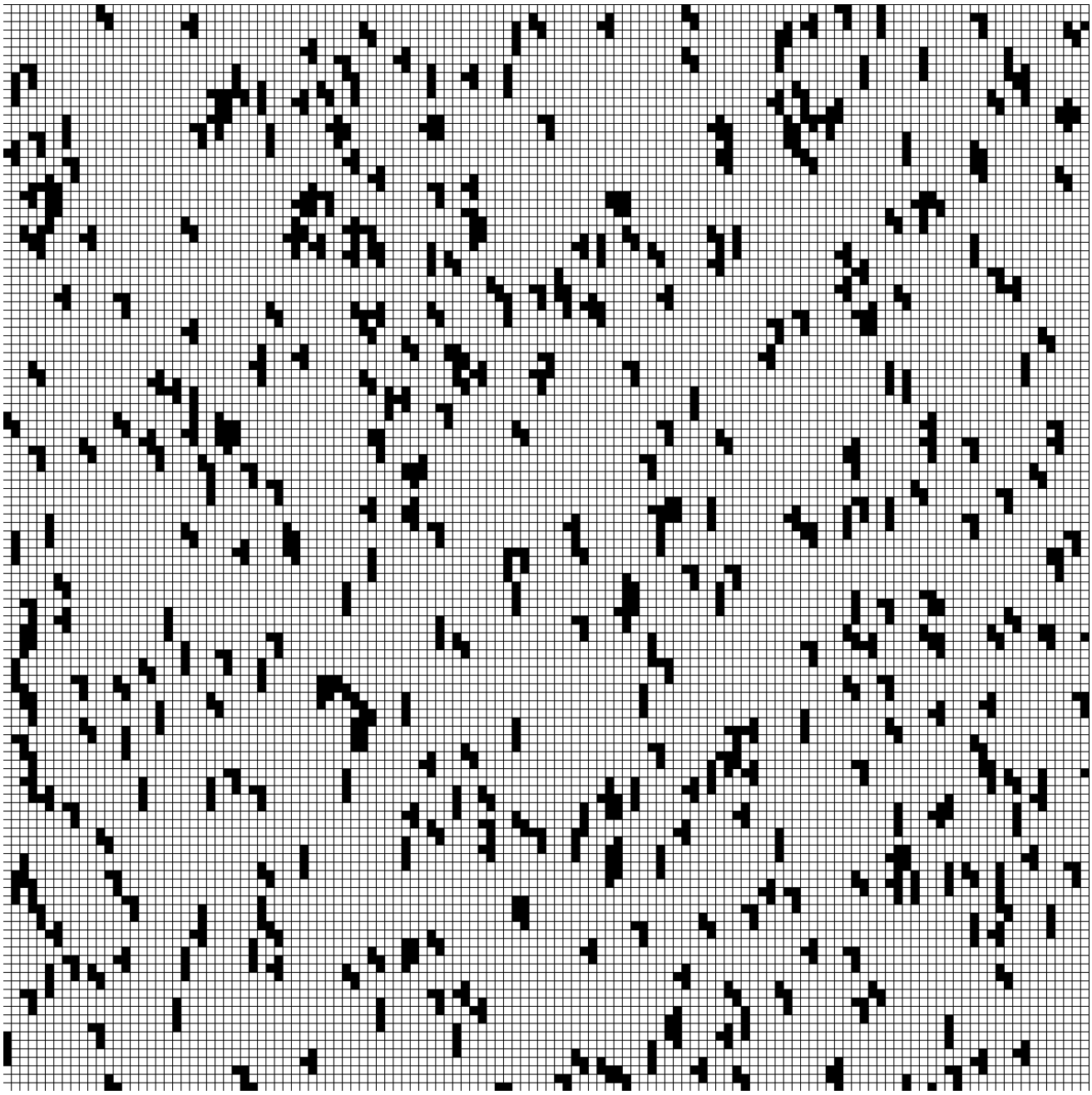
Sum the whole matrix looking for the '1' bit and totaling it. Returns the summation of the covered pixels on the grid array.

```
void plot_object(sf::RenderWindow *window)
```

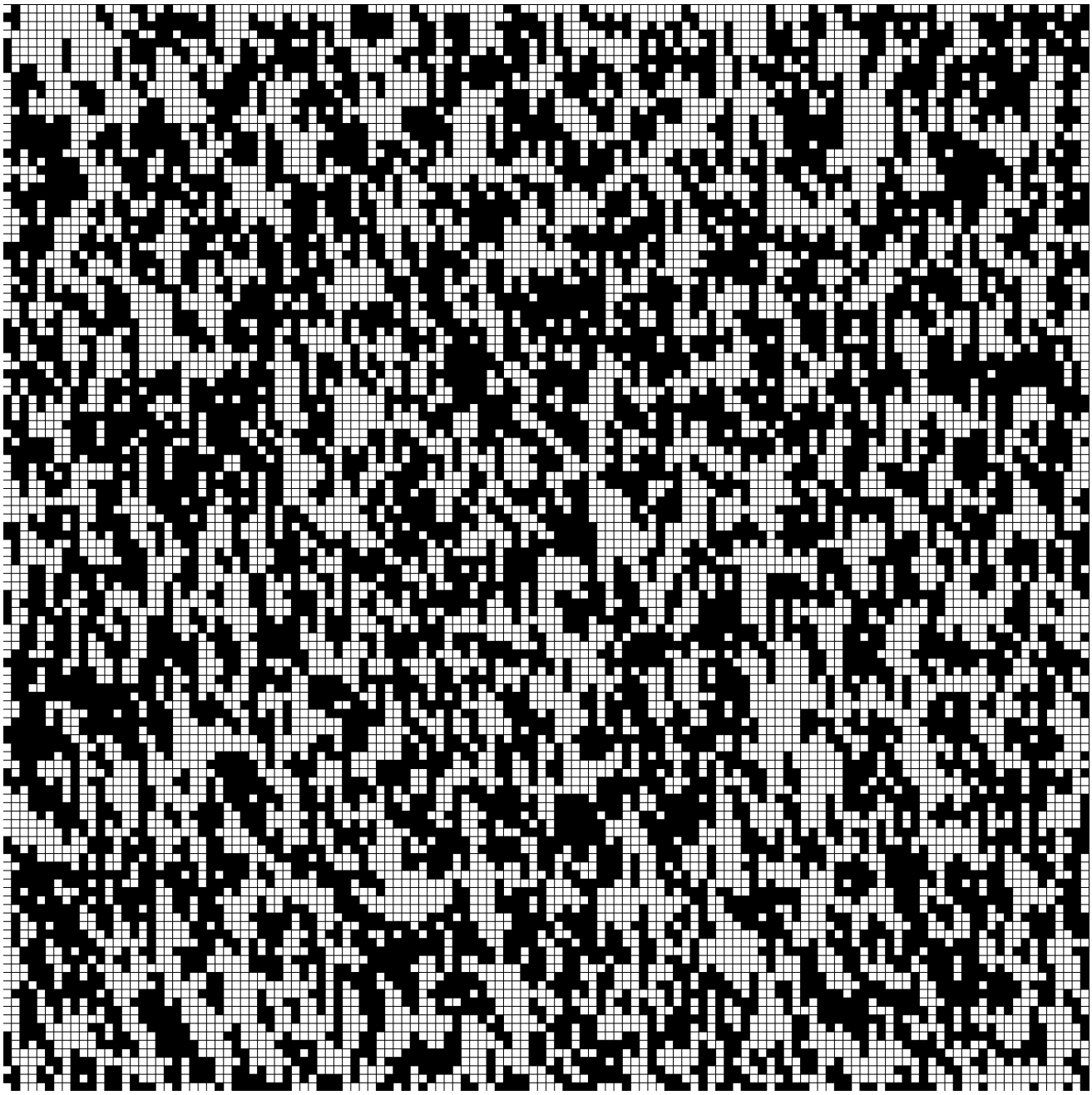
Function to execute sequence of statements to place the block on the window.

Results

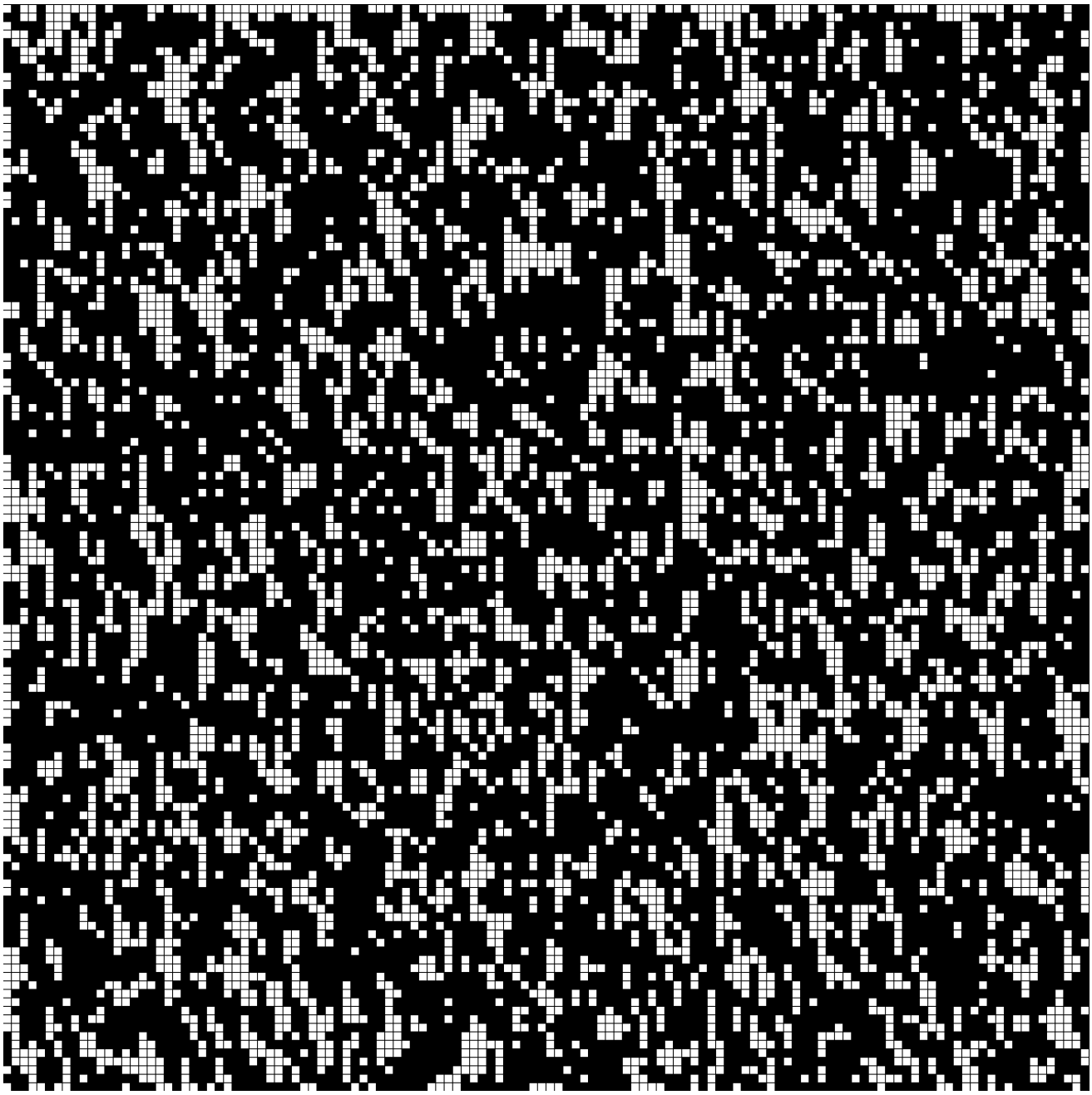
- **10% Coverage**:



- **50% Coverage**:



- **70% Coverage**:



Motion Planning Introduction

Find an interesting application of motion planning, and create a short report. Format may be written, slide presentation, video, or other. Cite references, and give clear examples. What do you find interesting about this method? What kind of problem does it solve? What familiar applications is it used in?

Be brief, consider this an elevator pitch to the rest of the class. Post your application summary, along with a short introduction about yourself, to the course discussion board.

Essay

The reference paper shows the work done by University of Pennsylvania to generate aggressive controllers for agile and quick maneuver in compact spaces. They have done tremendous work in developing and tuning the controllers super accurately. (*Though super lovely work there but we shall not be talking about it here.*)

The other aspect covered by the paper is Trajectory Generation via Sequential Composition. Sequential composition combines planning and control by computing a sequence of controllers to execute rather than a single trajectory, offering greater safety guarantees.

- Phase 1: Hover control (stiff) to a desired position
- Phase 2: 3D path following toward a desired position, r_L, and yaw angle, ψ_G
- Phase 3: Altitude control to desired pitch angle, θ_G , yaw angle, ψ_G, and zero roll

- Phase 4: Altitude control to zero pitch angle, yaw angle, ψ_G, and zero roll
- Phase 5: Hover control (soft) to a desired position.

This type of change in trajectories considering the motion type can help into smoother motions which consider the dynamics and the motion capabilities of the Robot.

Citation

Citation: Mellinger D, Michael N, Kumar V. Trajectory generation and control for precise aggressive maneuvers with quadrotors. The International Journal of Robotics Research. 2012;31(5):664-674. doi:10.1177/0278364911434236

Aggressive Maneuvers for Autonomous Quadrotor Flight: Reference Video

Submission

Parth Patel (he/him/his)

Wednesday

Hey there, my name is Parth Patel, you can contact me a pbpatel@wpi.edu. My undergraduate degree was in Electrical engineering, and I'm very interested to learn more about path planning. I like to do DIY projects in my free time, including Embedded Systems Design, 3D printing, and Programming. I also like to work out and develop the habit of reading a lot.

I would like to do a project focused on UAVs to implement on my own drone, but I'm open to other ideas in the field of mobile robots or manipulators.

Paper Discussion

The reference paper shows the work done by University of Pennsylvania to generate aggressive controllers for agile and quick maneuver in compact spaces. They have done tremendous work in developing and tuning the controllers super accurately. (*Though super lovely work there but we shall not be talking about it here.*)

The other aspect covered by the paper is Trajectory Generation via Sequential Composition. Sequential composition combines planning and control by computing a sequence of controllers to execute rather than a single trajectory, offering greater safety guarantees.

- Phase 1: Hover control (stiff) to a desired position
- Phase 2: 3D path following toward a desired position, $\mathrm{r\_\{L\}rL}$, and yaw angle, $\mathrm{\backslash psi\_\{G\}\psi G}$
- Phase 3: Altitude control to desired pitch angle, $\mathrm{\backslash theta\_\{G\}\theta G}$, yaw angle, $\mathrm{\backslash psi\_\{G\}\psi G}$, and zero roll
- Phase 4: Altitude control to zero pitch angle, yaw angle, $\mathrm{\backslash psi\_\{G\}\psi G}$, and zero roll
- Phase 5: Hover control (soft) to a desired position.

This type of change in trajectories considering the motion type can help into smoother motions which consider the dynamics and the motion capabilities of the Robot.

Citation

Citation: Mellinger D, Michael N, Kumar V. Trajectory generation and control for precise aggressive maneuvers with quadrotors. The International Journal of Robotics Research. 2012;31(5):664-674. doi:10.1177/0278364911434236

Aggressive Maneuvers for Autonomous Quadrotor Flight:
Reference Video

Edited by Parth Patel on Jan 13 at 12:10am