Rover

VILNIUS UNIVERSITY, FACULTY OF MATHEMATICS AND INFORMATICS, INFORMATION TECHNOLOGIES STUDY PROGRAMME

Done by:

Gabrielius Drungilas

Aistė Grigalūnaitė

Nedas Janušauskas

Adomas Jonavičius

Supervisor: Linas Bukauskas

Outline

- Project introduction
- Updates
- New functionality
- Current status

Project introduction

The goal of the project is to design an autonomous robot, which would be able to map, detect and avoid objects in an indoor environment.

Updates

- Accelerometer/Gyroscope functionality
- Increased current in motors
- Sensing system

Updates: sensing system

Before:

- Distance is calculated on call
- Value calculated on single measurement
- Readings can be unreliable

After:

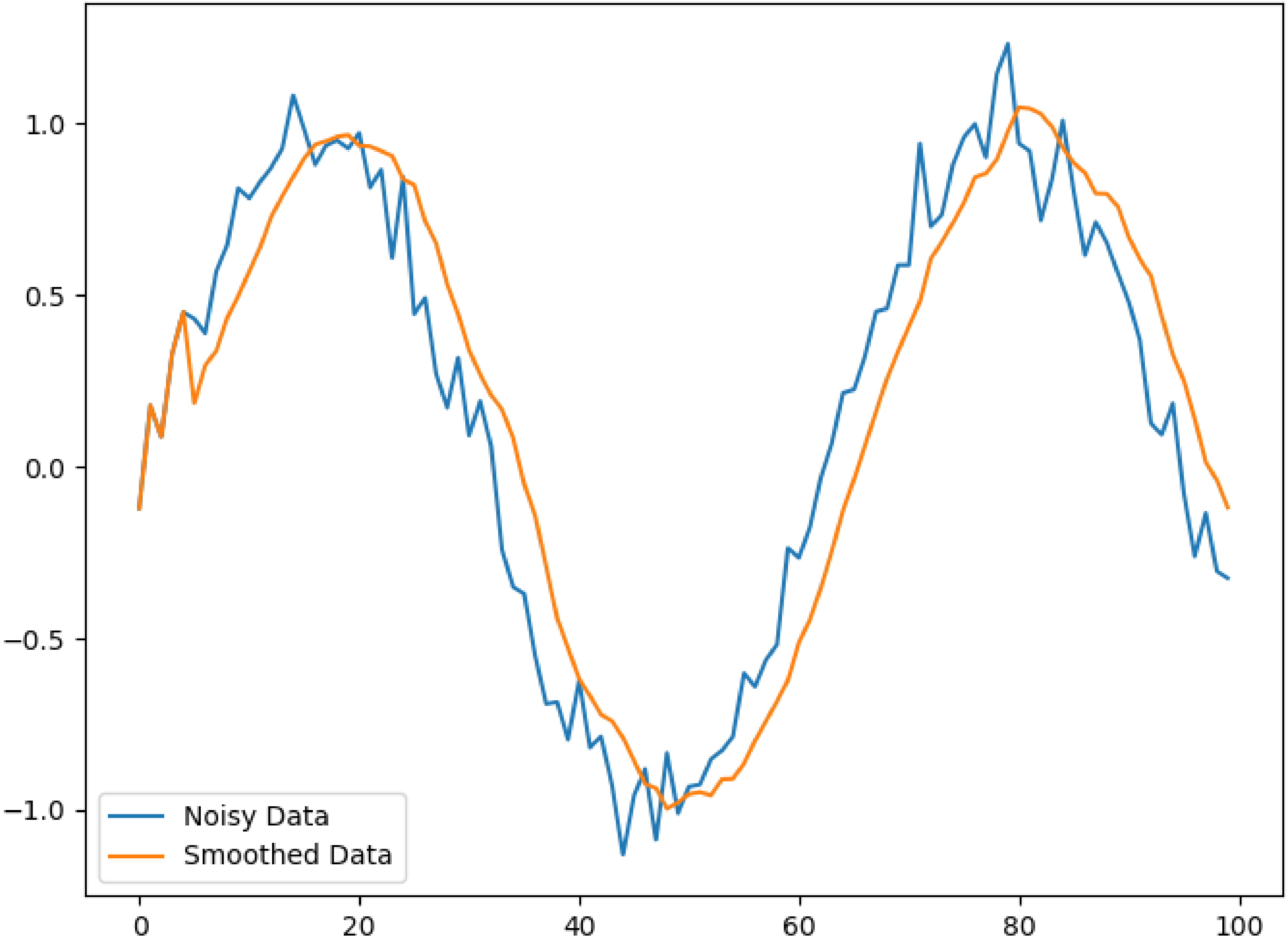
- Distance is calculated constantly
- Value is calculated as moving average of measurements
- Reading are more reliable

Updates: sensing system

```
set array to [0, 0, 0, 0, 0]

function update_array(value)
    shift all elements in array by one element to the left
    set last element in array to value

function get_distance()
    return average(array)
```


1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Noisy Data
Smoothed Data

New functionality

- Path-finding algorithm
- Exploring algorithm

New functionality: Path-finding

A pathfinding algorithm to find the shortest path between the start and end on a map.

Algorithm:

Recursively calculates the distances from the start tile to all adjacent tiles eventually reaching all reachable tiles on the map.

If the end is reachable, we add tiles to the path and get the next tile by checking the neighboring tile's distances to start.

In the end, we reach the start and reverse path to get the correct order.

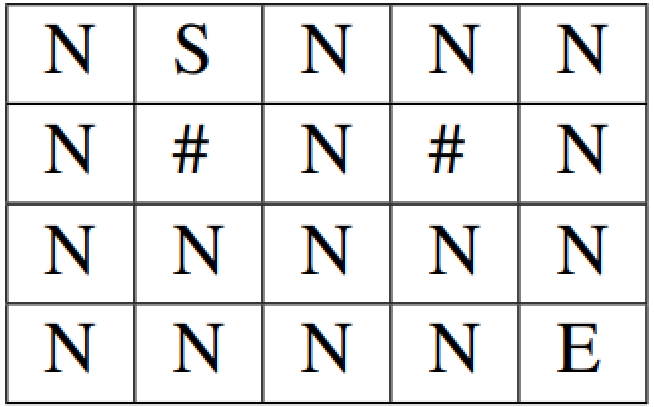
New functionality: Path-finding

```
Function get_shortest_path() returns List[Tile]:
    path = []
    if start is not set or end is not set:
        return path
    start_tile = map[start]
    end_tile = map[end]
    start_tile.distance = 0
    calculate_distances(start_tile)
    if end_tile.distance is None:
        return path
    cur_tile = end_tile
    for _ in range(cur_tile.distance):
        path.append(cur_tile)
        cur_tile = get_lowest_distance_neighbour(cur_tile)
    return reversed path
```

New functionality: Path-finding

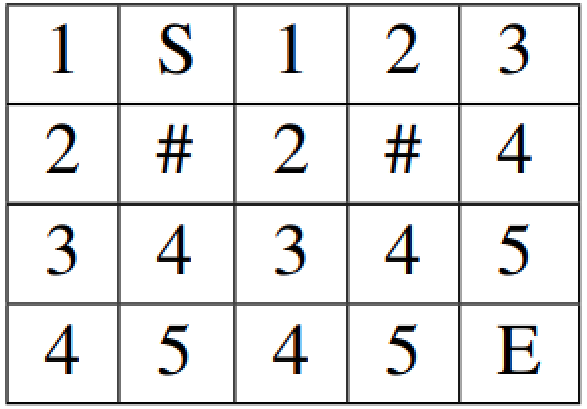
Example:
S - start, E - end, # - obstacle, N – distance is None, number – distance from the start

Before:



N	S	N	N	N
N	#	N	#	N
N	N	N	N	N
N	N	N	N	E

After:



1	S	1	2	3
2	#	2	#	4
3	4	3	4	5
4	5	4	5	E

New functionality: Exploring

Built the code base for further development of exploring tactics.

2 types of implemented exploring tactics:

- Ride while possible, if not turn in random available direction.
- Ride straight until robot finds wall, then ride among the wall, following its every turn.

New functionality: Exploring

Real map of first tactic:

```
  N N
  N N
  N#NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN#
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNN                                                                                             NNN
  NNN                                                                                             NNN
  NNN                                                                                             #NN
  NNN                                                                                             #NN
  NNN                                                                                             #NN
  NNN
  NNN
  NNN
  NNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN#####
 N#NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNS
```

Current status

Total hours spent on project: 632

Current tasks:

- Thoroughly test new exploring tactics.
- Build more complex, fool proof exploring tactics.
- Reposition distance sensors for better visibility.

THANK YOU FOR YOUR ATTENTION

Extra

```
Function calculate_distances(Tile tile, int n=0) returns None:
    neighbours = get_neighbours(tile)
    for i in range(len(neighbours)):
        if neighbours[i] is an obstacle or is unknown:
            continue
        elif neighbours[i].distance is None or neighbours[i].distance > n + 1:
            neighbours[i].distance = n + 1
            if neighbours[i] is the end:
                break
            calculate_distances(neighbours[i], n + 1)
```