Assignment 4: Cutting grass

Part a)

We know from the map that at from **i = 0 to i = 21, j = 0 to j = 17** there is a wall that cannot be crossed and that this wall divides the map area into two. Besides, when the rain value reaches above a certain value, the robot cannot cut grass. The most important thing is that the battery should not run out of power.

In order to remove grass from a wider area, I take a row-by-row approach to path planning. When the rain is greater than 0.2 it will stop in that area and cut when the length of the grass in that area is greater than 0.2. Here's how I implemented this function.

First I defined a couple of global variables.

pos_i and **pos_j** : record real-time location

myI and **myJ** : Sensor position reading

Grass_Centre : length of grass

Battery_bot : amount of power

map_rain : amount of rain from sensor

charge_time : number of charges

bool rt = 0 : whether to find last return point

actions : robot commands

bool wall: determining if it is behind a wall

```
uint32_t pos_i;
uint32_t pos_j;
uint32_t myI;
uint32_t myJ;
float Grass_Centre;
float Battery_bot;
float map_rain;
int actions;
int findmyway;
int charge_times;
bool rt = 0;
bool wall = 0;
```

1. gotoChargeStation behaviour

This layer is mainly about finding the path to charge station.First, I defined the behaviour of going home called ***gotoChargeStation***. When the robot is on the other side of the wall, if robot i is positioned to the

left of the wall, then the robot needs to go right before returning, otherwise it can go straight back up. When the robot is in any other general position, the robot returns by going up and then to the left.

When the robot returns to the charging station, the robot is stopped. Finally record the first charge.

```
auto gotoChargeStation(uint32_t i, uint32_t j){
  if(j >= 18){
    if(i < 22){
      actions = 4;
      }
    else{
        actions = 2;
      }
    }
  else{
    if(j > 0){
        actions = 2;
      }
    else if(i > 0){
        actions = 8;
      }
    }

    if(i == 0 && j == 0){
    actions = 0;


  } //new
  charge_times = 1;
  }
```

2. gotoCut behaviour

This layer is for the cutting grass function. As I choose to cut the grass row by row, from the map we can know that in the odd rows the bots need to go right and in the even rows the bots go left.

First I define a function to determine whether the robot is behind a wall, and if the robot is behind a wall, it goes left and then right.

When the robot is in the row of the wall, when i=23, the last frame of the wall, the robot goes down and thus achieves crossing over the wall and into the space behind it. I set the battery threshold to 0.4 and inside this layer recorded the robot's position when the battery is at 0.4.

```
void behindwall(uint32_t i){
  if (i != 0){
    actions = 8;
  } else {
    actions = 0;
    wall = 1;
  }
}
```

```
auto gotoCut(uint32_t i, uint32_t j, float battery){

  if(j==18 && wall==0){
    behindwall(i);
  }
  else if(j%2 == 0){
    if(i == 39){
      actions = 6;
    } else {
      actions = 4;
    }
  } else if(j == 17){
    if(i == 23){
      actions = 6;
    }else{
      actions = 8;
    }
  } else {
    if(i == 0){
      actions = 6;
    }else{
      actions = 8;
    }
  }


  if(battery >= 0.4f){
    pos_i = i;
    pos_j = j;
  }


}
```

3. cutGrass behaviour

In this layer, the robot is in charge of cutting the grass. In this layer I have set the robot to move when the rain is less than 0.2 and to cut the grass so that its length is 0.2. When the rain is greater than 0.2, the robot stops running.

```
void cutGrass(uint32_t i, uint32_t j, float grasscentre, float battery, float Rain){
  if(grasscentre > 0.2f && Rain < 0.2f){
    actions = 0;
  } else {
    gotoCut(i, j, battery);
  }

  if(Rain > 0.2){
    actions = 0;
  }
```

4. find behaviour

On this layer, it is used to return the robot to the end of its last point home. When the robot is behind a wall, the robot first travels to the right so that its position **i** is greater than the length of the wall, then goes down and finally returns to the left to the last return point. When the robot is not behind the wall, the robot travels right and then down to return to the last point. When the robot returns to the last point, the robot stops and returns rt = 0, indicating that it has finished searching for the last point.

```
void find(uint32_t i, uint32_t j){

  if (pos_i <= 21 && pos_j >= 18) {
    if (i < 22 && j < pos_j) {
        actions = 4;
    }
    else if (j < pos_j) {
        actions = 6;
    }
    else if(i > pos_i){
        actions = 8;
    }
  }
  else{
    if (i < pos_i) {
      actions = 4;
    }
    else if(j < pos_j){
      actions = 6;
    }

  }

  if(i==pos_i && j==pos_j){
    rt = 0;
    actions = 0;
  }

}
```

In the main program, we use these behaviours.

The sensors first read the location of the robot in the map, the length of the grass, the battery level and the amount of rain.

When the robot's battery is less than 0.4, the robot returns to the charging station for recharging (**gotoChargeStation behaviour**). When the robot is at the charging station and the battery level is less than 1, charging is performed. When charging is complete, it detects if it is the first charge, if it is the first charge, it is given to find the last point rt = 1 and carry out the robot's return operation (**find behaviour**), otherwise the robot starts to move to the right from the starting point. When the robot starts to move and battery level is large than 0.4, the robot cuts the grass (**cutGrass behaviour**).

```
auto msg = cluon::extractMessage<tme290::grass::Sensors>(
    std::move(envelope));
    Battery_bot = msg.battery();
    myI = msg.i();
    myJ = msg.j();
    Grass_Centre = msg.grassCentre();
    map_rain = msg.rain();

    if(Battery_bot < 0.4f){
      gotoChargeStation(myI, myJ);
    } else if(Battery_bot < 1.0f && myI == 0 && myJ == 0){
      actions = 0;
    } else if(Battery_bot > 0.99f && myI == 0 && myJ == 0){
      if(charge_times == 1){
        rt = 1;
        find(myI, myJ);
      }else{
        std::cout<< "actions 4" << std::endl;
      actions = 4;
      }
    } else if((myI != 0 || myJ != 0) && Battery_bot > 0.4){
      if(rt == 1){
        std::cout << "here rt" << rt << std::endl;
        find(myI, myJ);

      }
      else{
       cutGrass(myI,myJ, Grass_Centre, Battery_bot, map_rain);
      }
    }
```

You can run the following command:

Run it first

docker run -ti --rm --net=host -e COLUMNS=$COLUMNS -e LINES=$LINES -e TERM=$TERM olbender/tme290-grass-simulator-amd64:v0.0.7 tme290-sim-grass --cid=111 --time-limit=0 --dt=0.1 –verbose

then run

mkdir b

cd b

cmake ..

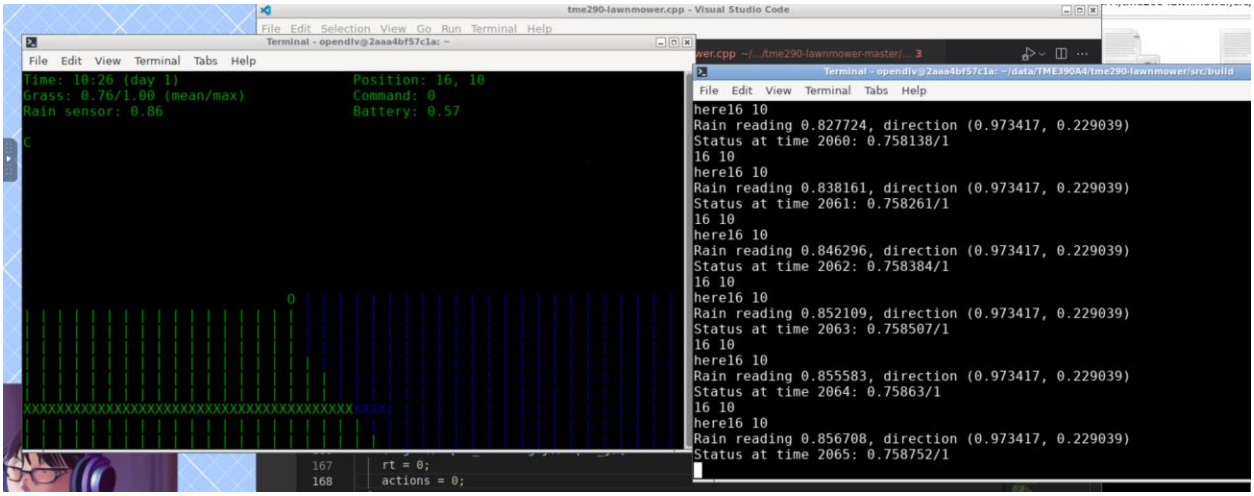
make

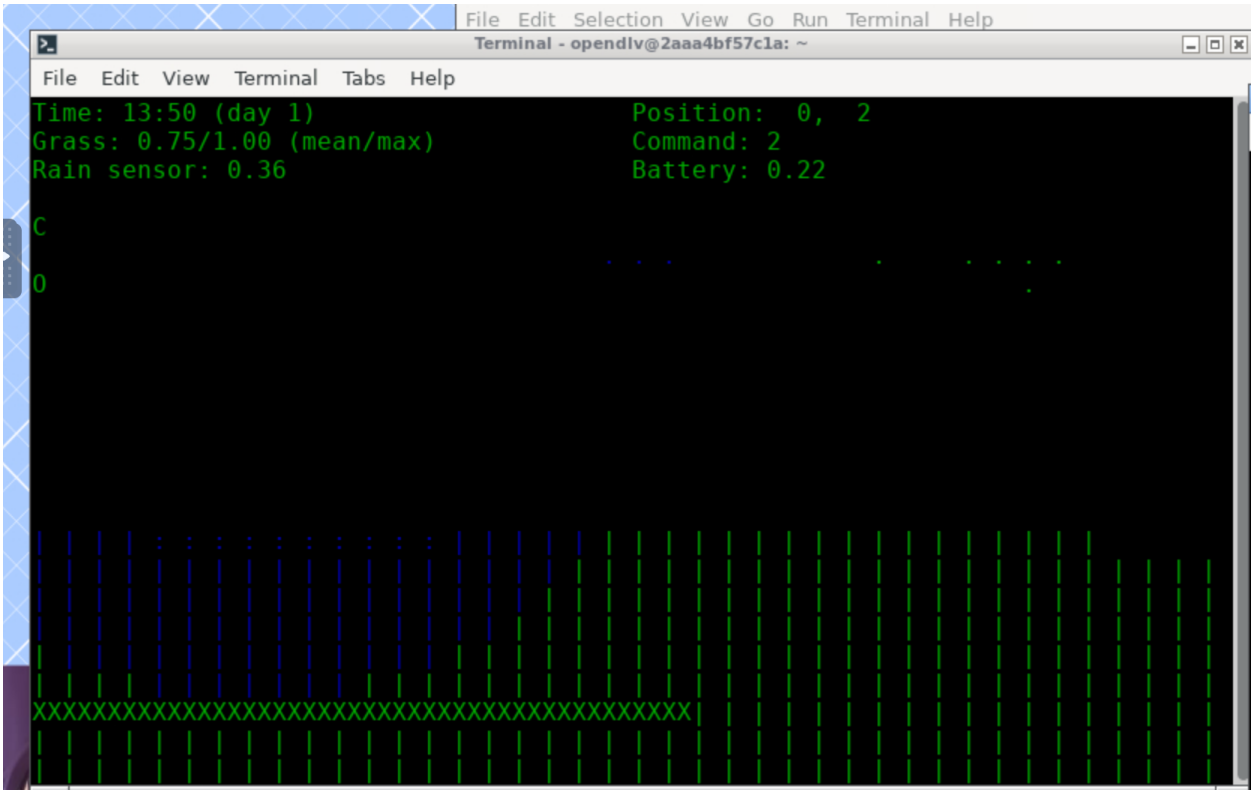
finally run:

./tme290-lawnmower --cid=111 --verbose

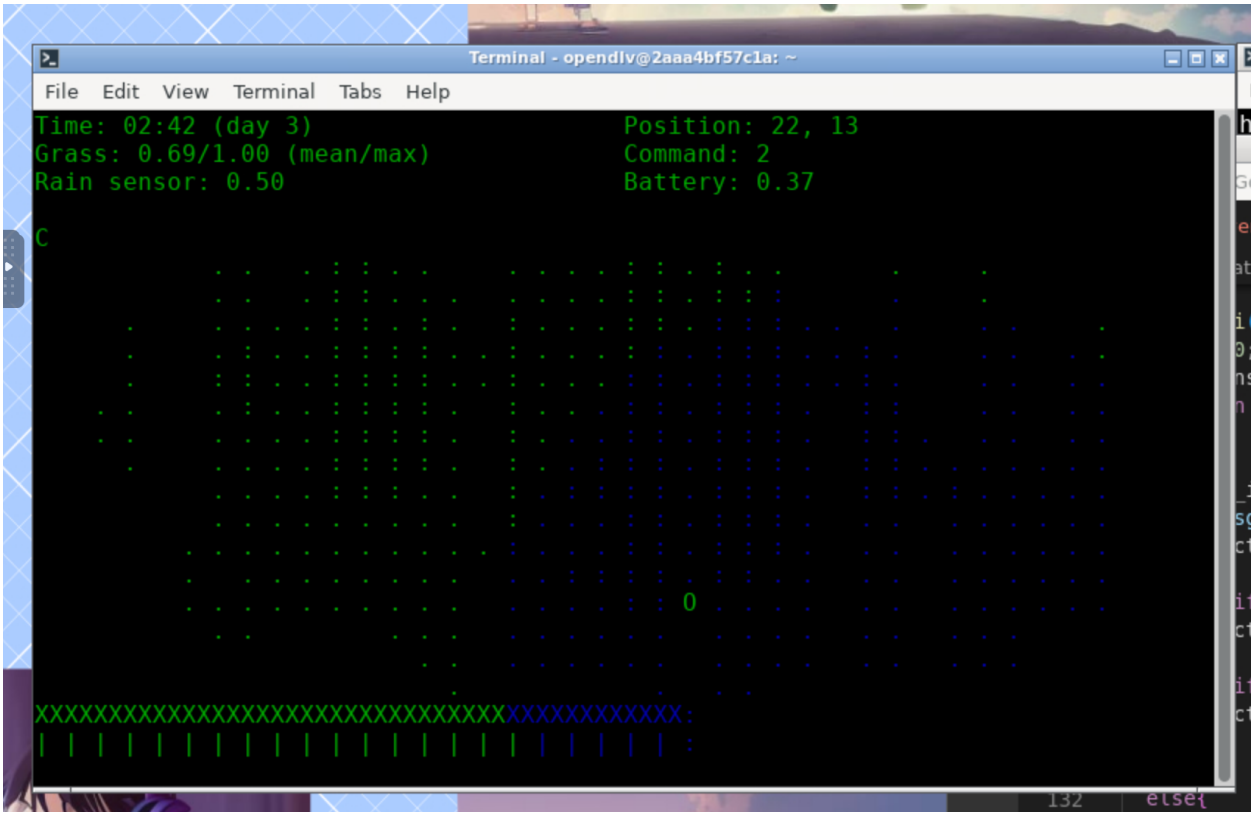
The robot's performance is shown as below:

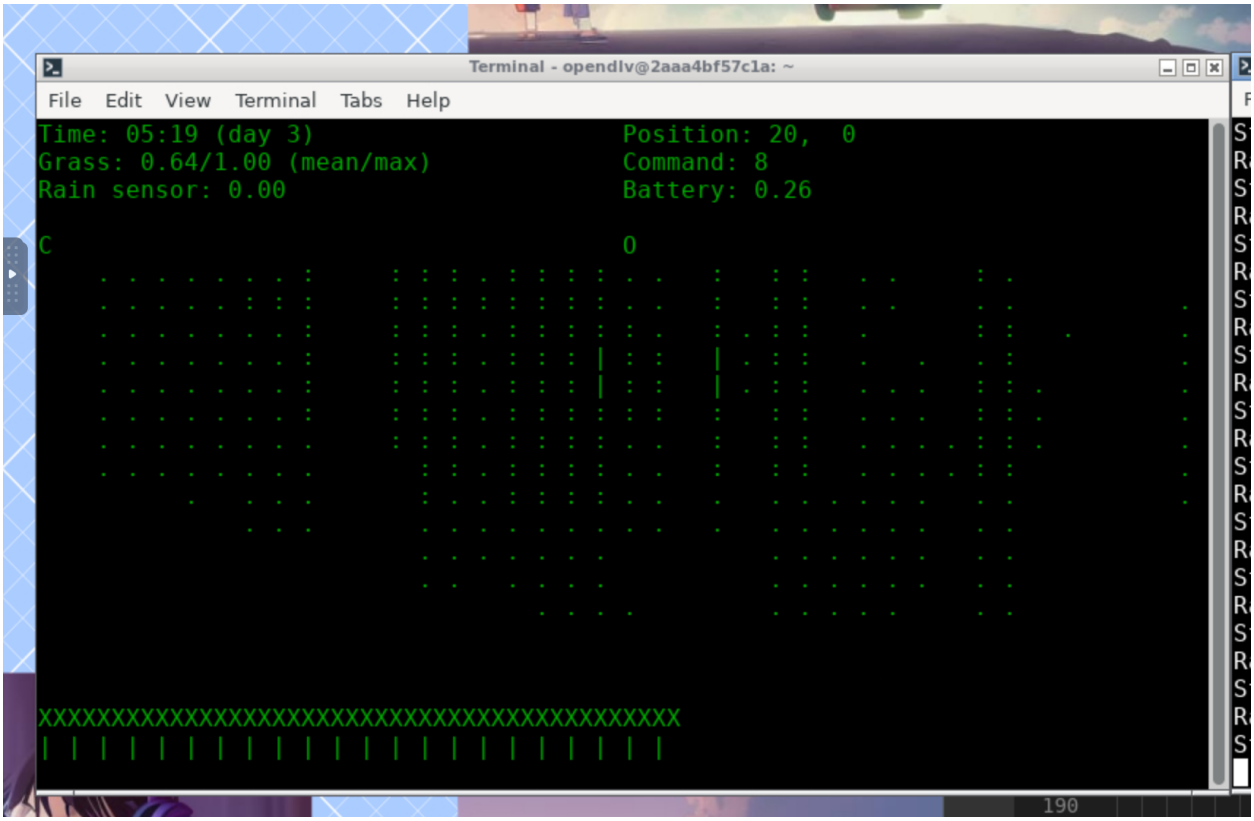
In front of the wall:



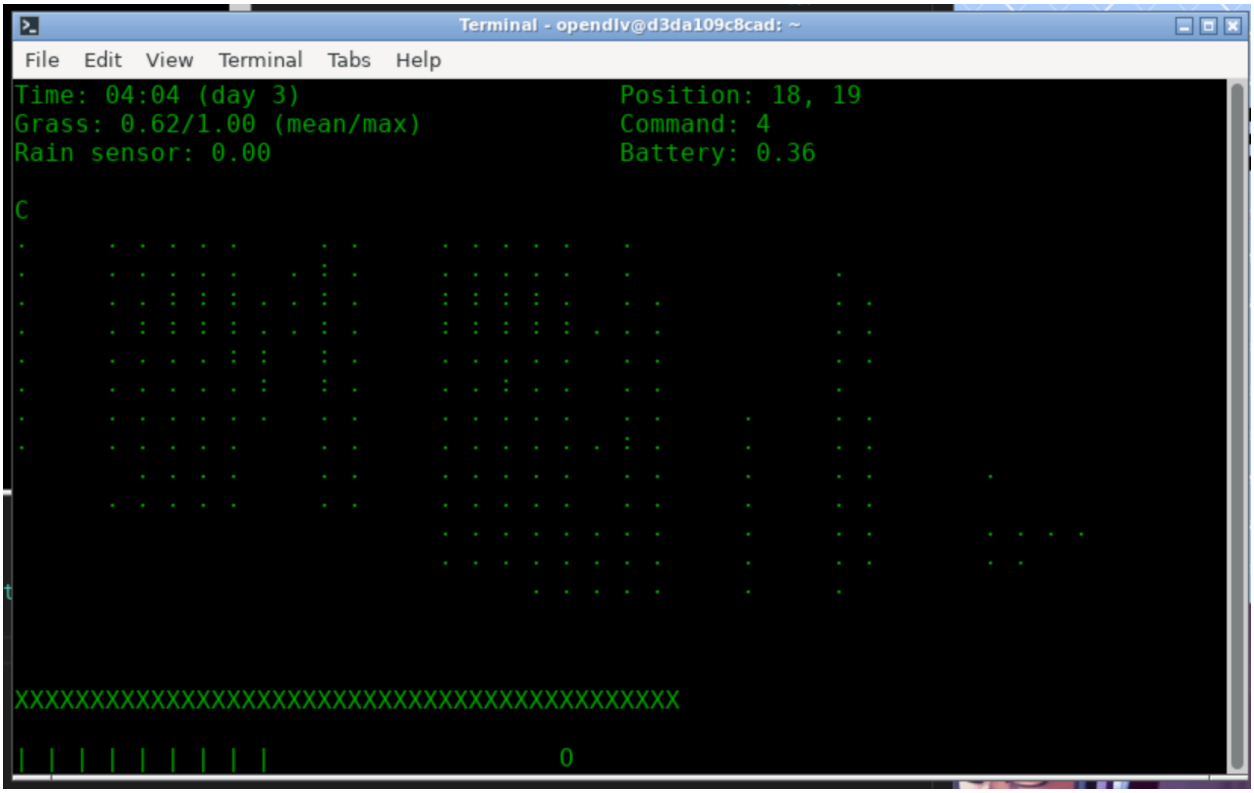


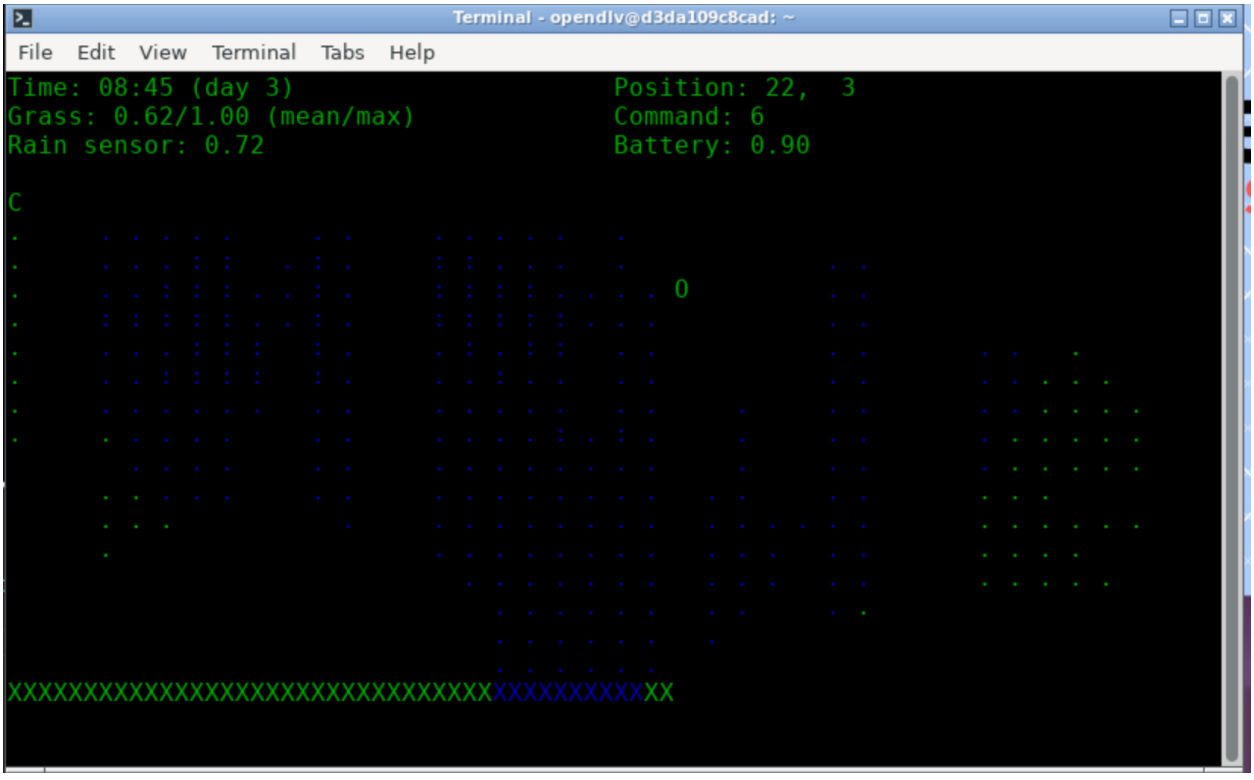
Stop when raining:





Behind the wall:


Terminal - opendlv@d3da109c8cad: ~
File Edit View Terminal Tabs Help
Time: 04:04 (day 3)
Grass: 0.62/1.00 (mean/max)
Rain sensor: 0.00
Position: 18, 19
Command: 4
Battery: 0.36
C
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
0


Terminal - opendlv@d3da109c8cad: ~
File Edit View Terminal Tabs Help
Time: 08:45 (day 3)
Grass: 0.62/1.00 (mean/max)
Rain sensor: 0.72
Position: 22, 3
Command: 6
Battery: 0.90
C
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX