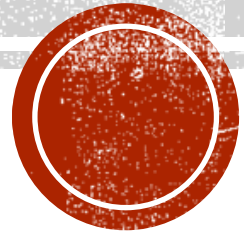
MARS ROVER

Project Supervisions:

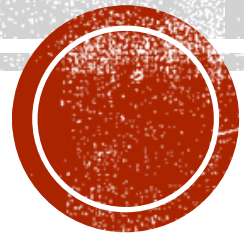
Prof. Gamal M. El-Bayoumi Prof. Ayman H. Kassem



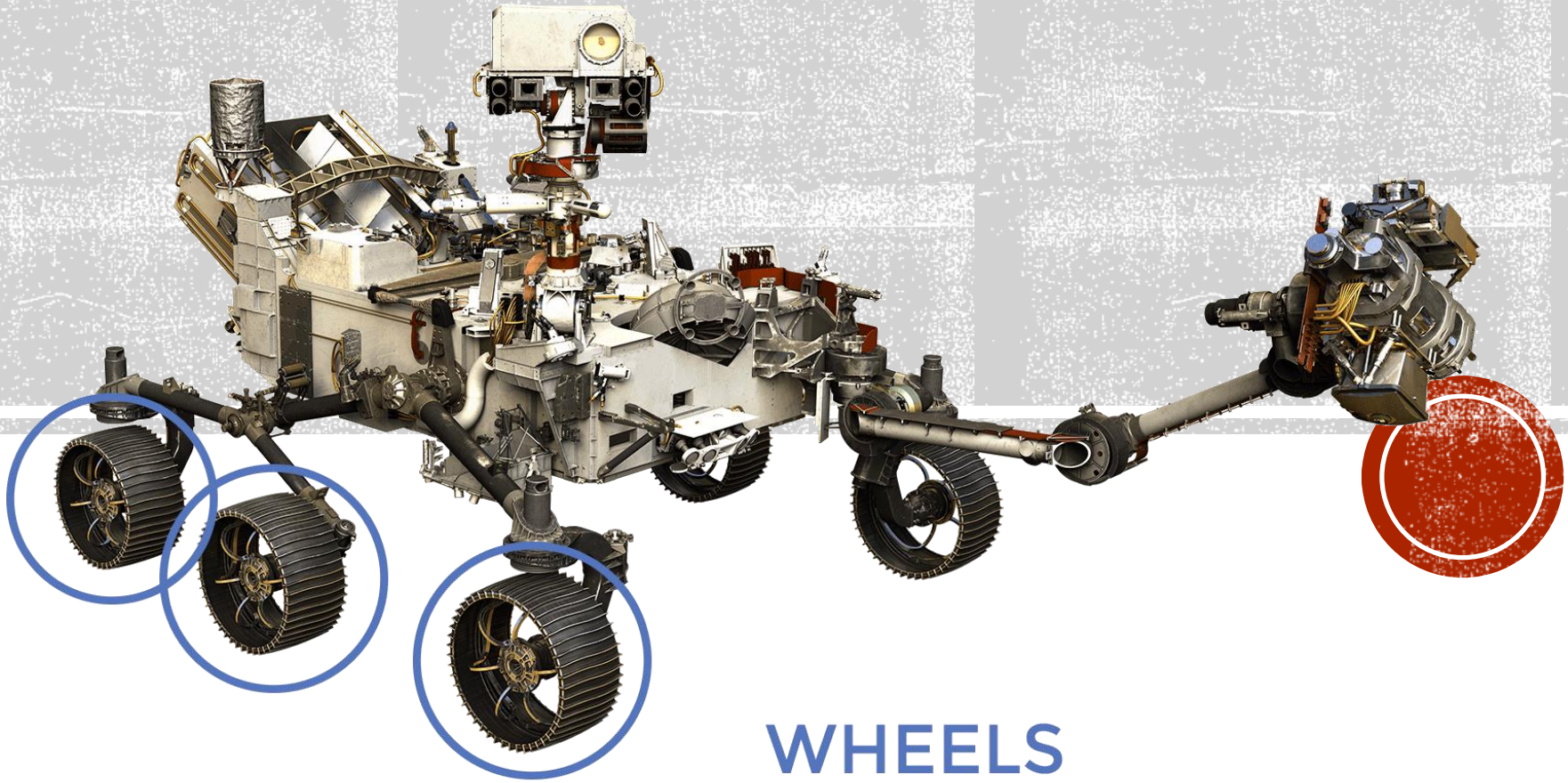
TEAM MEMBERS

- Karim Ahmed Kamal El-din
- Mohamed Ismail Ibrahim
- Anas Abd-Elmenam Farrag

INTRODUCTION



MARS ROVER



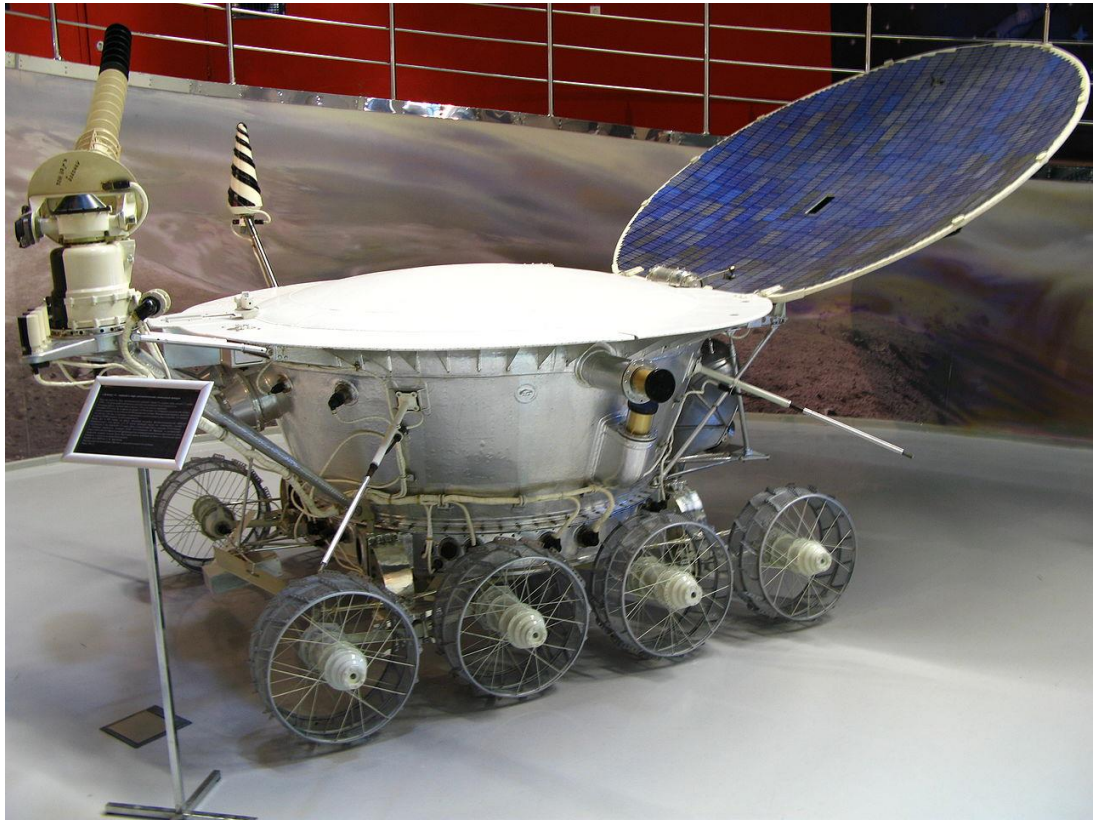
WHEELS

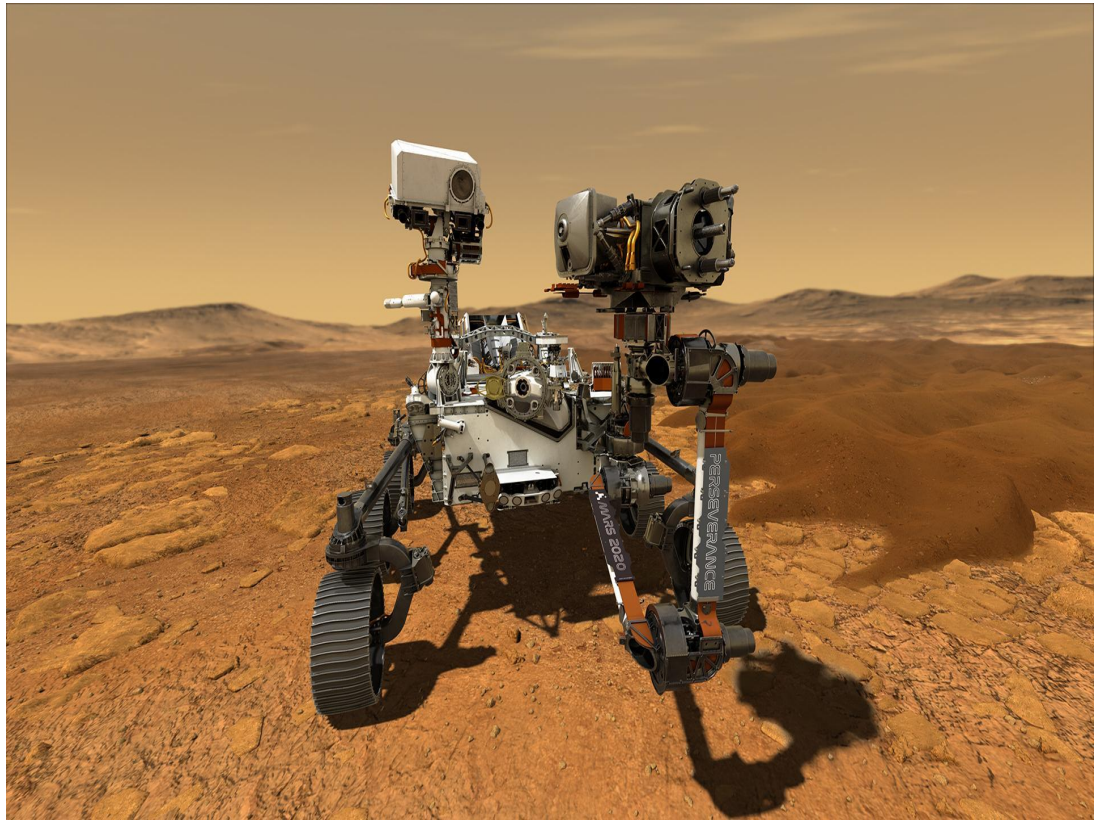
- A **Mars rover** is a motor vehicle that travels across the surface of the planet Mars upon arrival.
- They examine more territory, they can be directed to interesting features, they can place themselves in sunny positions to weather winter months, and they can advance the knowledge of how to perform very remote robotic vehicle control.
- They have a lot of sensors for identifying what’s Around the Mars
- They have a robotic Arm can dig the ground or hold the Rockets

WHY ROVERS ?!

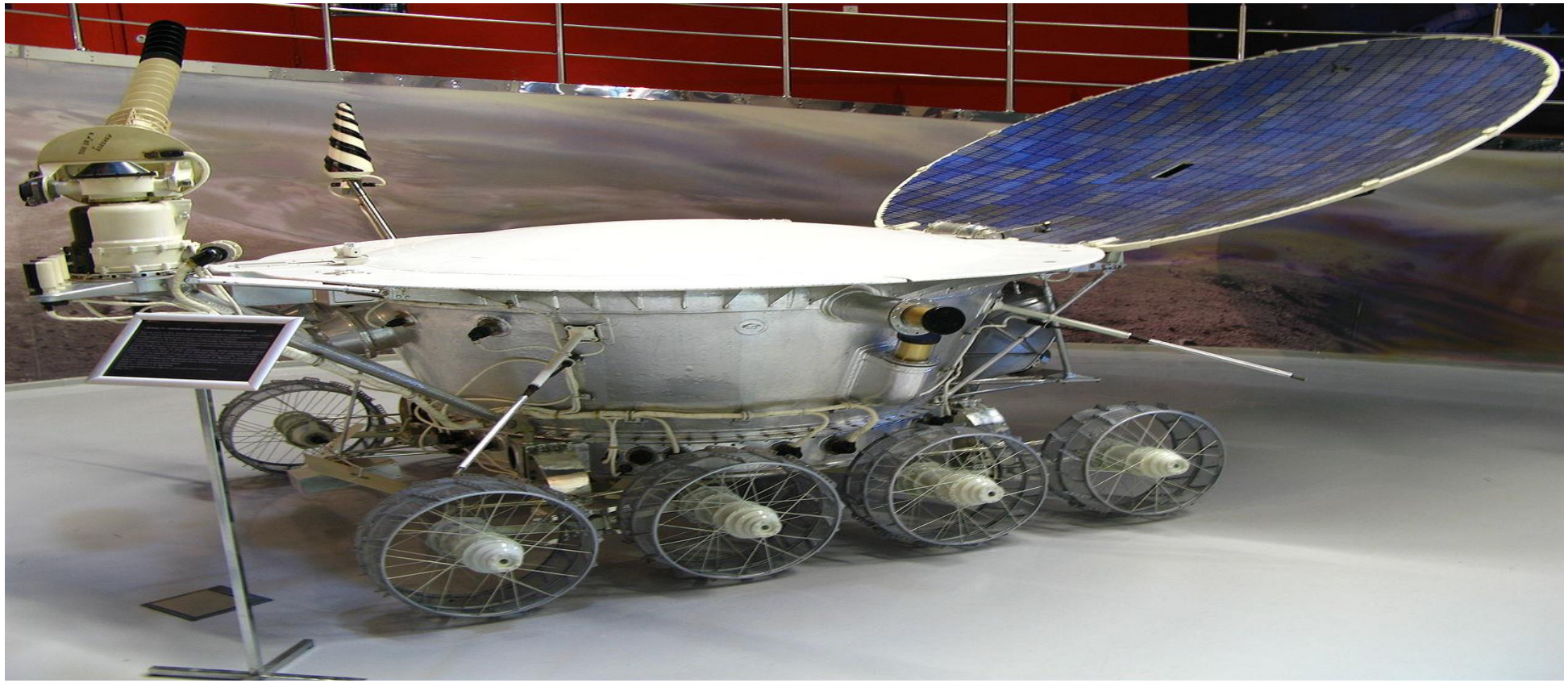
- Discovering the space without the risk of humans life
- Determine whether life ever arose on mars
- Characterize the climate of Mars
- Saving time
- Reduce the Cost

DEVELOPMENT OF THE ROVER





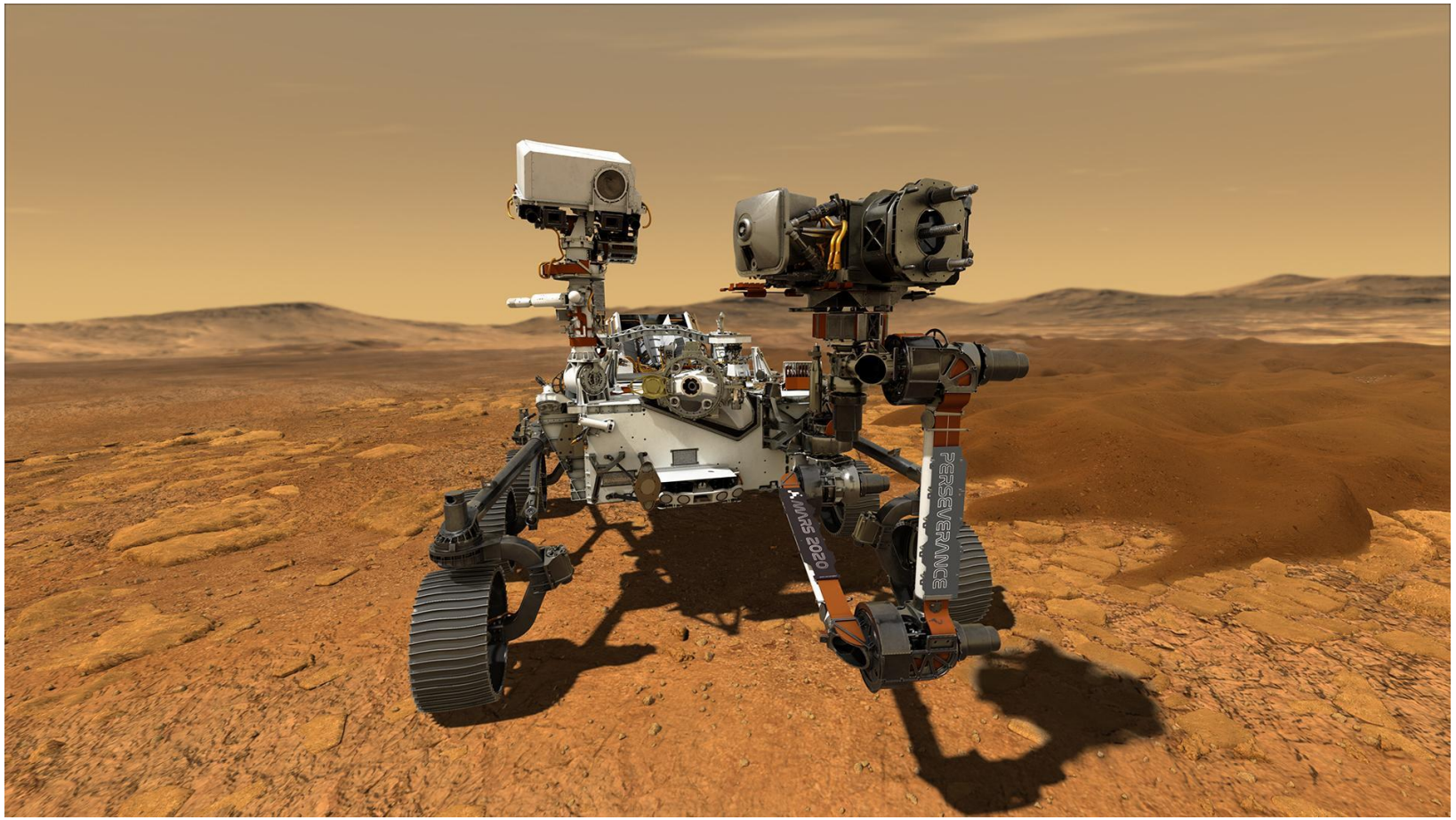
LUNOKHOD 1(1970)



LUNOKHOD 1

- 8 Wheels
- Cone Shaped Antenna
- Four Television Cameras
- Test Lunar Soil
- During 5 Days , The rover drove 197 m , and returned 14 close-up images
- After 10 months , They lost the Communication with Rover

PERSEVERANCE



PERSEVERANCE(MARS2020)

- **Launched on** July 30, 2020
- Expected to land on mars on February 18, 2021
- Mission duration at least one mars year(687 earth day)
- 1025 kg
- 7 scientific instruments
- Explore a geologically different landing site

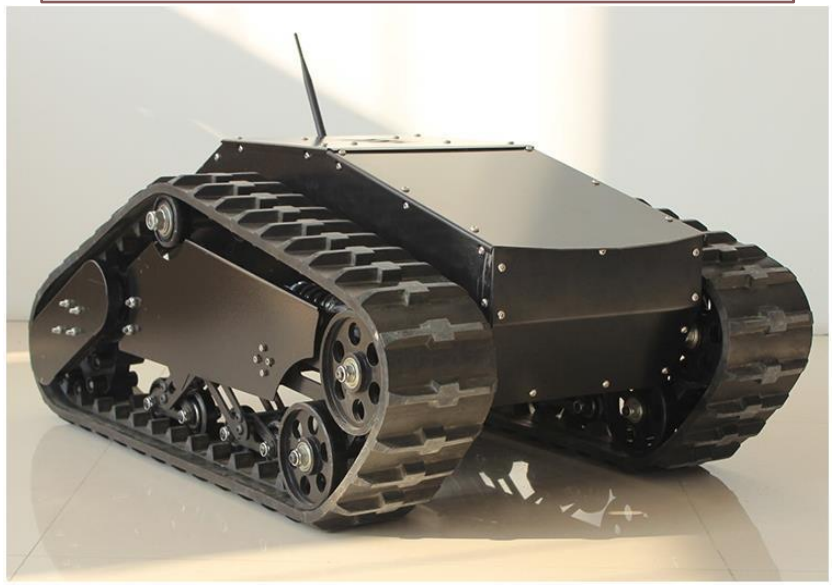
MECHANICAL SYSTEMS

- Locomotion
- Suspension
- Steering

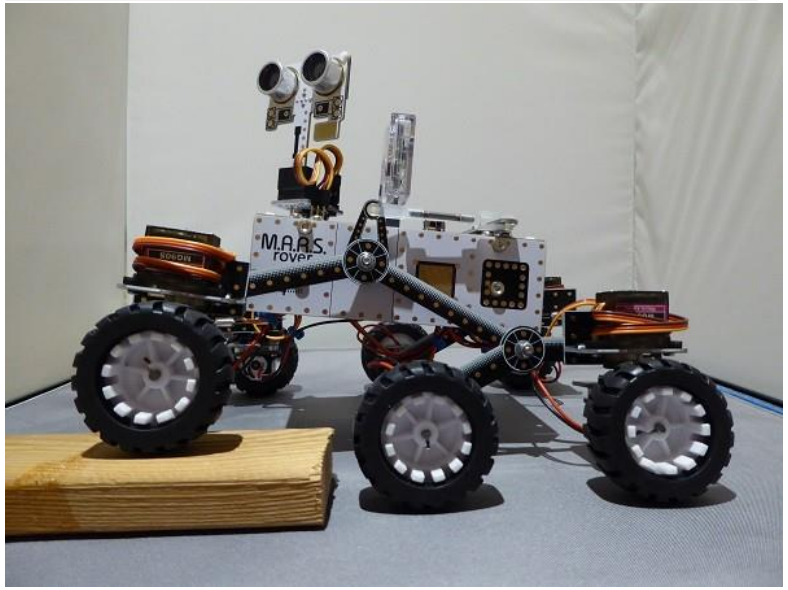
LOCOMOTION

- locomotion system which determines the stability and capacity while traversing on rough terrain

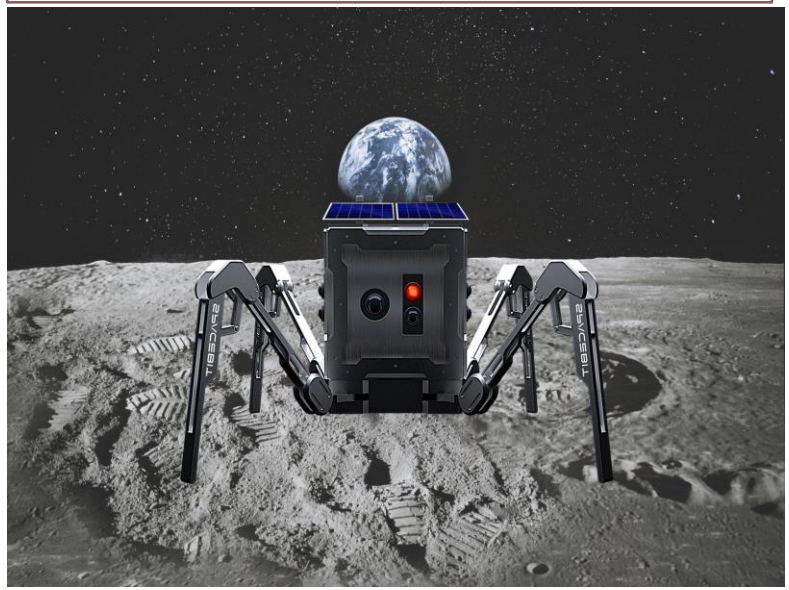
Tracked Rover



Wheeled Rover



Legged- Rover



LOCOMOTION

Previous Wheels (12 cm)

Our design wheels(radius = 25 cm)

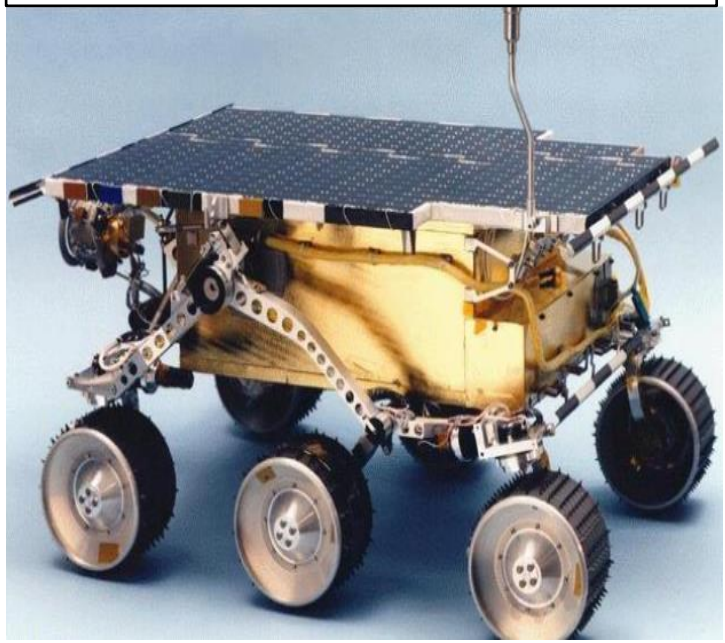
SUSPENSION SYSTEM

- which connects the wheels to the main body or platform

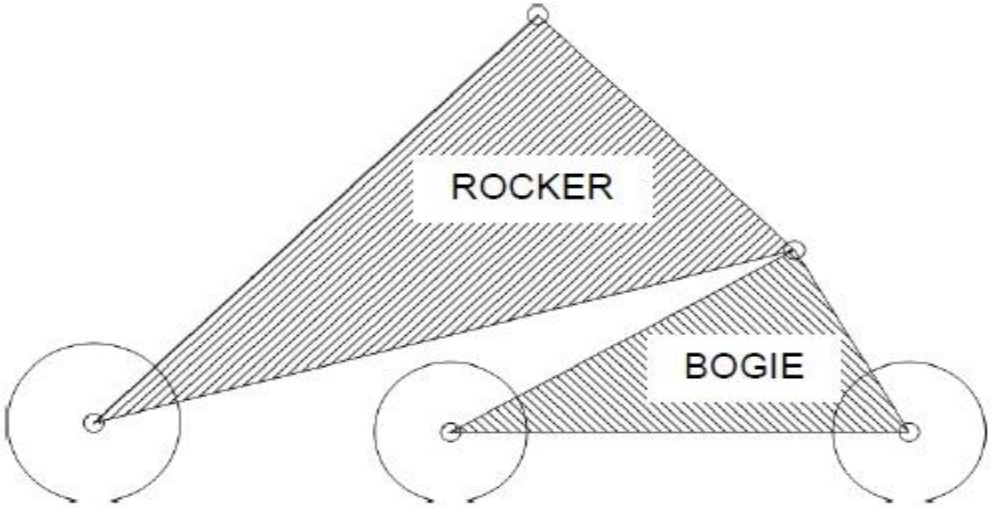
Independent Spring Suspension

Articulated Body Suspension

Rocker-Bogie Suspension



ROCKER-BOGIE SUSPENSION



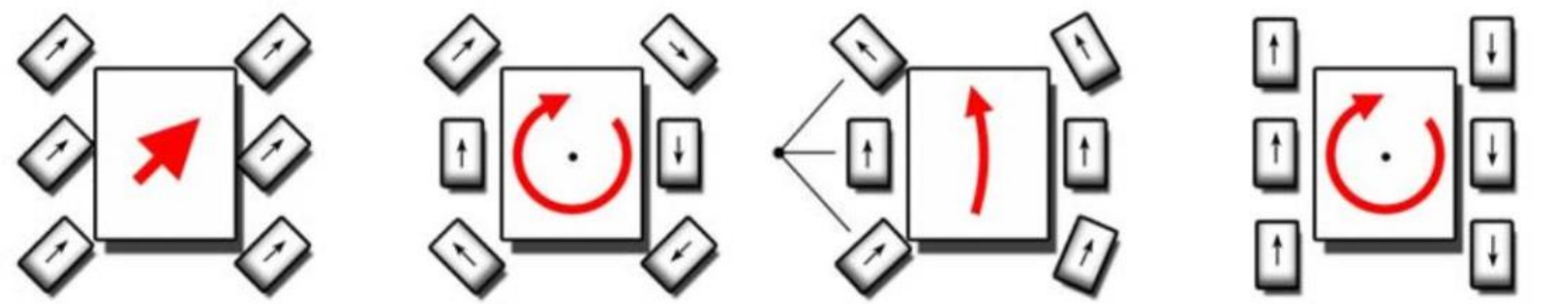
Rocker :Connects the front wheel to the differential and the bogie in the rear.

Bogie: Connects the middle and rear wheels to the rocker.

Differential: Connects to the left and right rockers and, to the rover body

The rocker bogie suspension is capable of a high degree of mobility. It has a ground clearance larger than a wheel diameter, unlike articulated body vehicles. The single rigid body is more stable for sensor mounting and thermal control

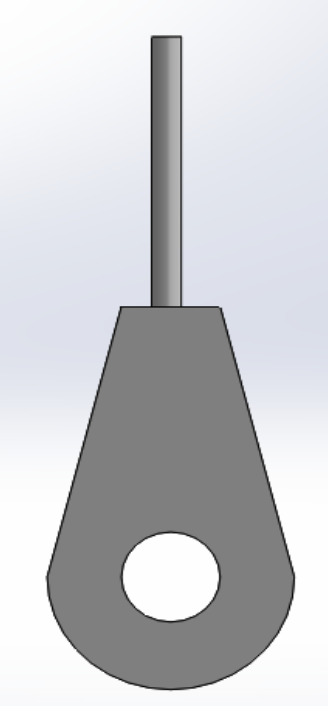
ALSO SUPPORTS THE DIFFERENT TYPES OF STEERINGS

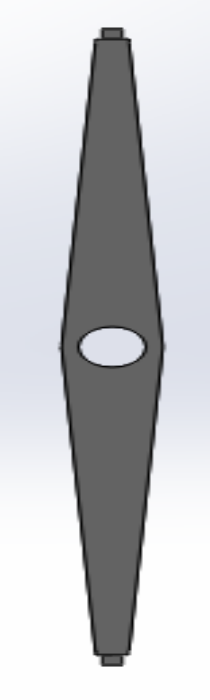


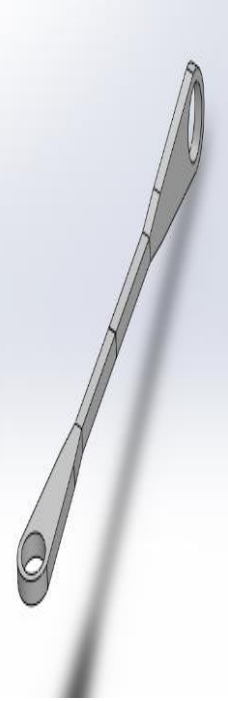
(a) *Crabbing* (b) *Zero radius* (c) *Ackerman* (d) *Differential/Skid*

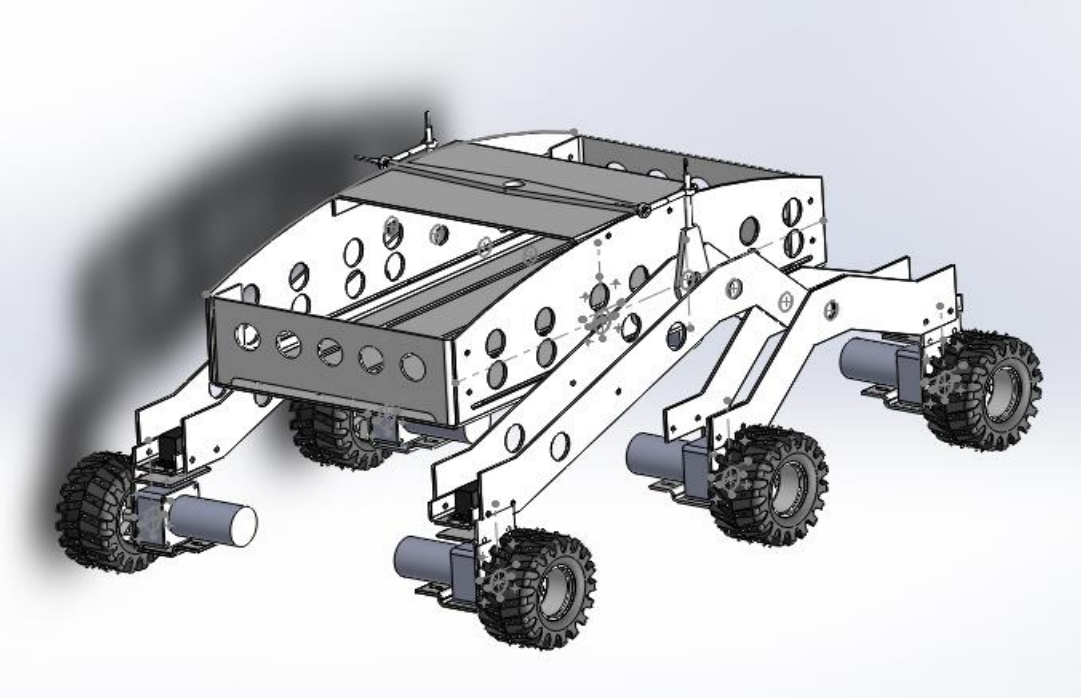
PREVIOUS DIFFERENTIAL PART(PIVOT)

1- can’t put the solar panels and the sensors
2- less stable in overcoming the terrain
3- has a problem in simulations (gazebo)



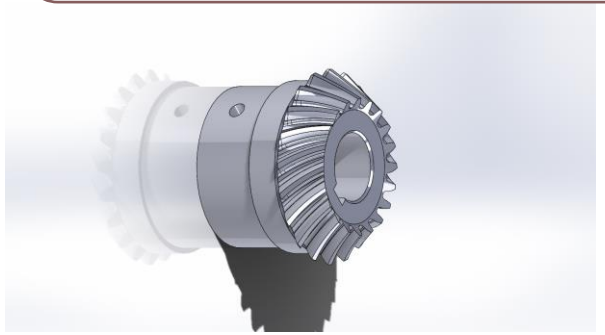


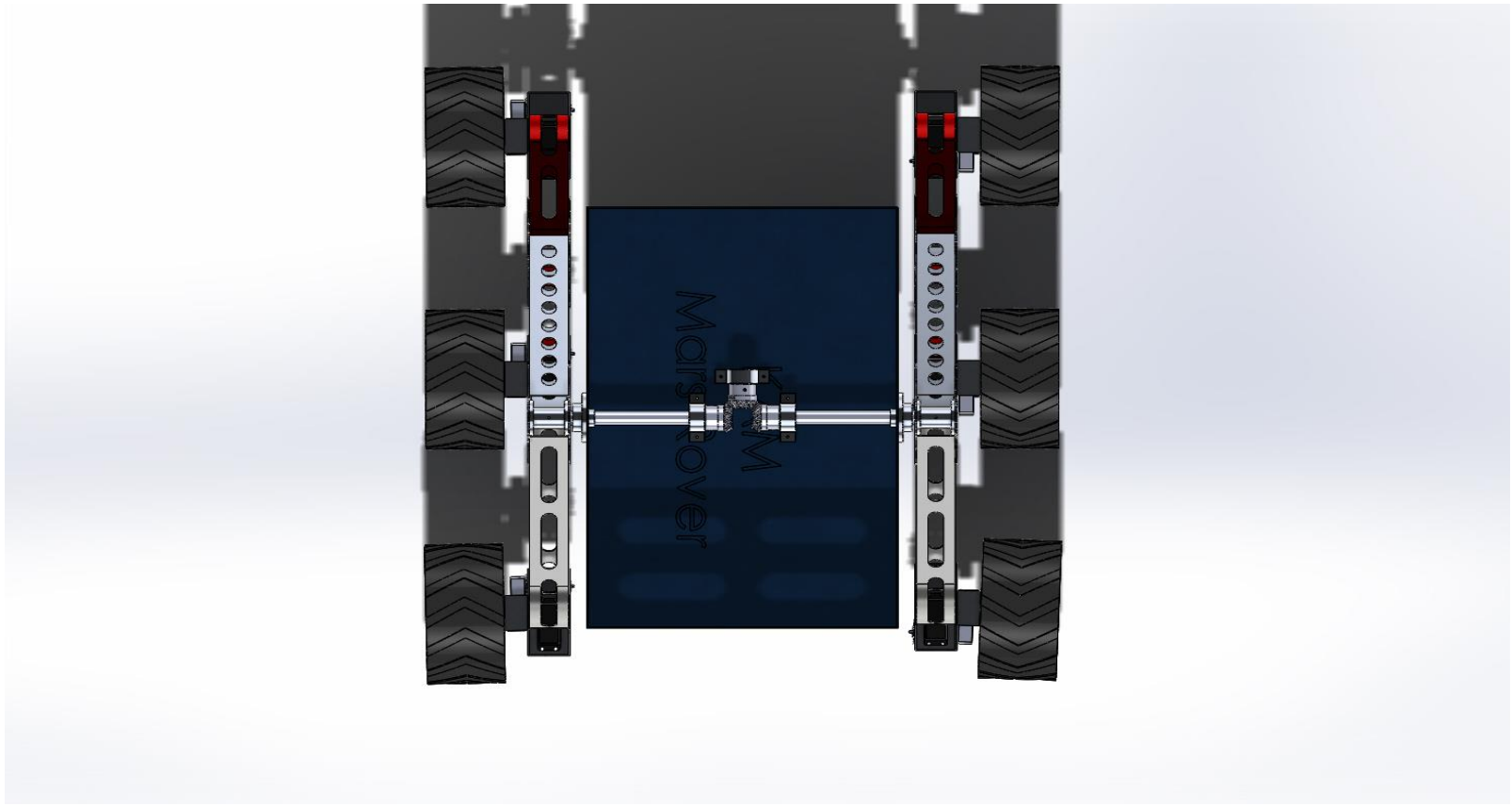




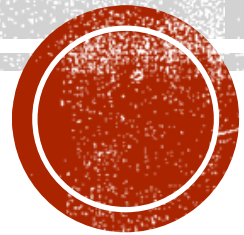
OUR DIFFERENTIAL DESIGN(DIFFERENTIAL GEAR)

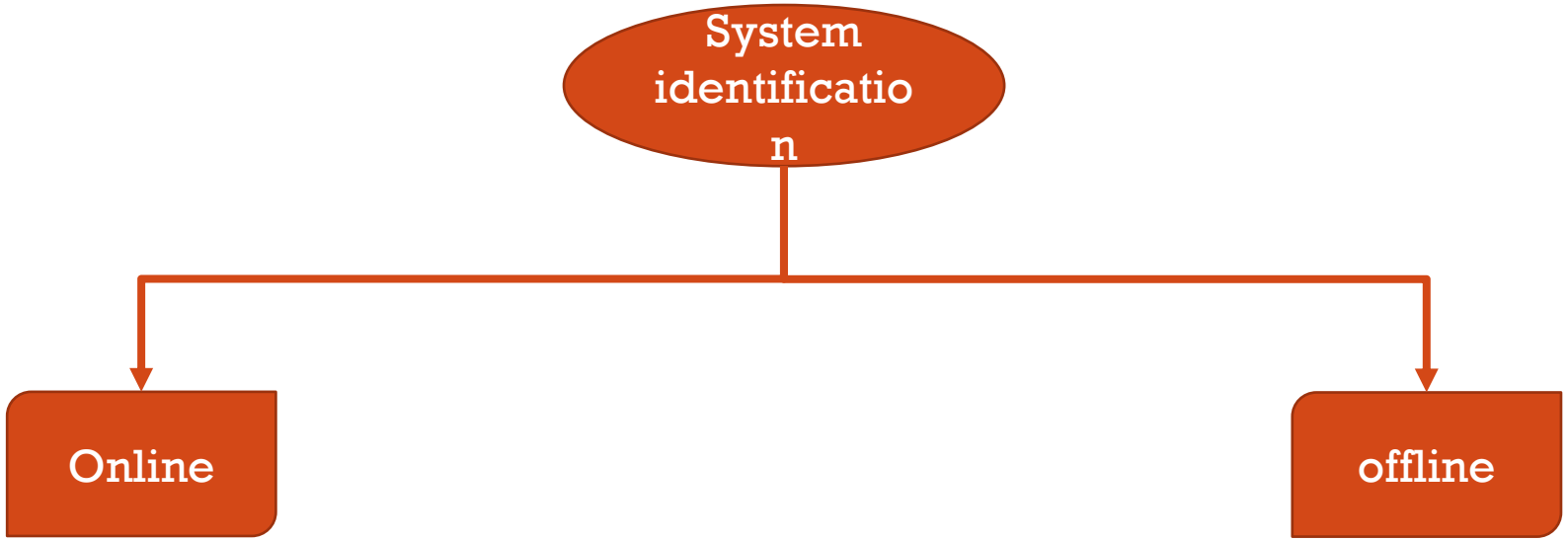
The differential is composed of three identical beveled gears situated 90-degrees from each other at the center of the rover





SYSTEM IDENTIFICATION





- **Online system identification** means that while rover operation system identification tool collects required data and expect the transfer function

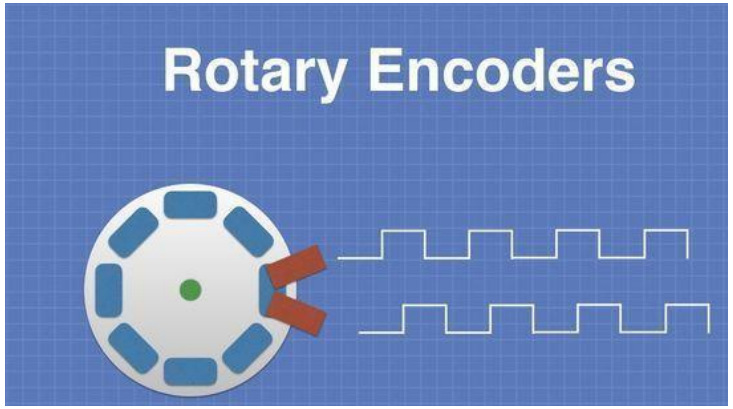
- **Offline system identification** means that we took required data then used the system identification on another time

MOTOR SPECS

12 V DC geared motor: 5000 rpm reduced to 83 rpm with reduction ratio 60.24

Encoder specs

Rotary encoder have 7 disk spots equipped with magnetic hall sensor(for each cycle has 7 high digital signals)



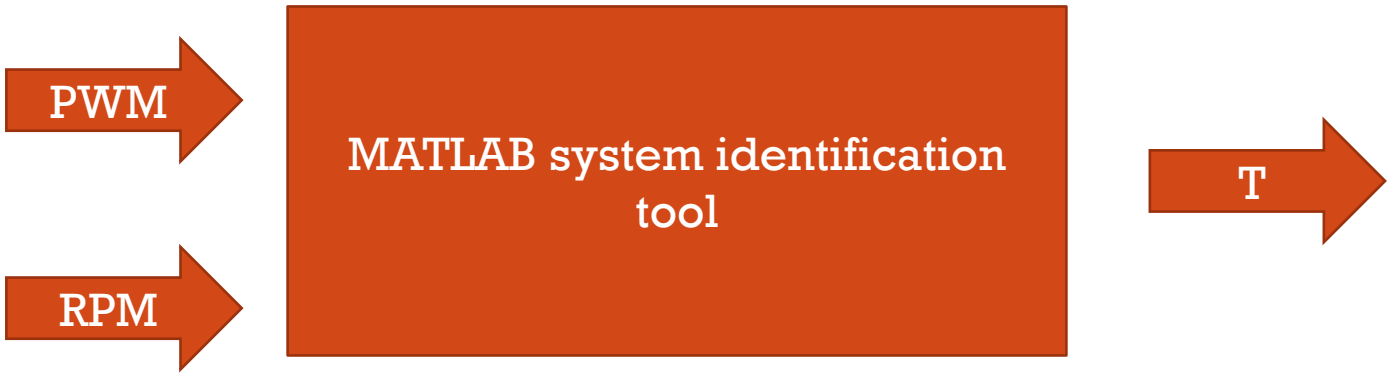
PWM

RPM

$$PWM = 255 * |\sin(\omega t)|$$

$$PWM = 255 * |\sin(2\pi f * no.of\ iterations * T_{sampling})|$$

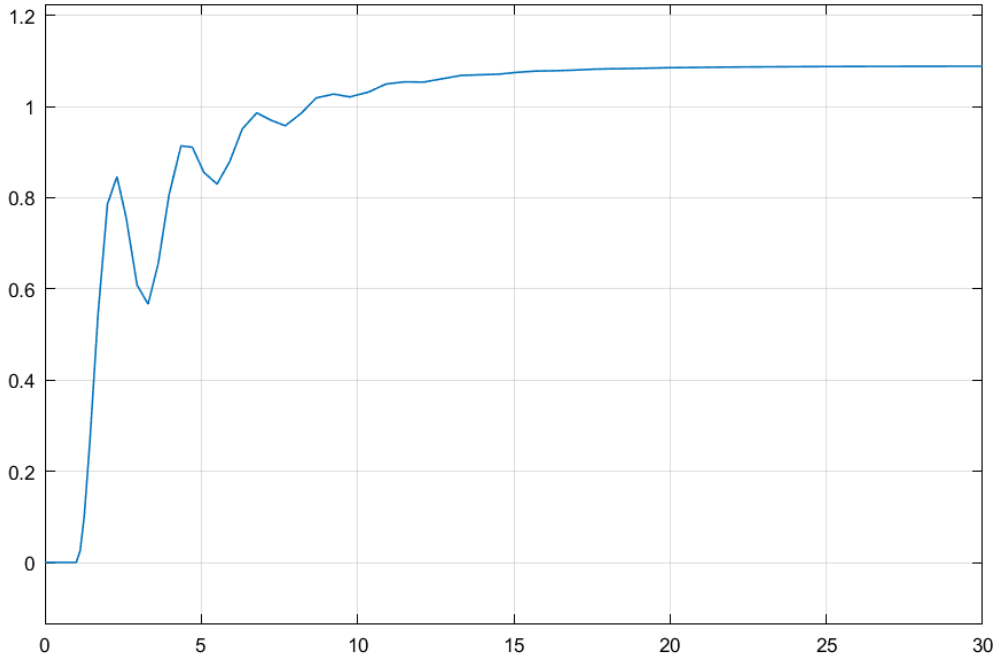
$$RPM = \frac{encoder\ count}{time\ in\ min} * \frac{1}{encoder\ disk\ spots\ * gear\ ratio}$$



Using this tool to all six motors getting the following results

$$T_1 = \frac{RPM}{PWM} = \frac{3.32\,s^2 + 2.431}{S^3 + 1.117\,s^2 + 8.276\,s + 2.233}$$

Response



CONTROLLER DESIGN "PID TUNER"

we perform Simulink model to simulate the transfer function in closed loop form with unity feed-back and PID controller

We perform 2 controllers showing the results in following figure PI and PID

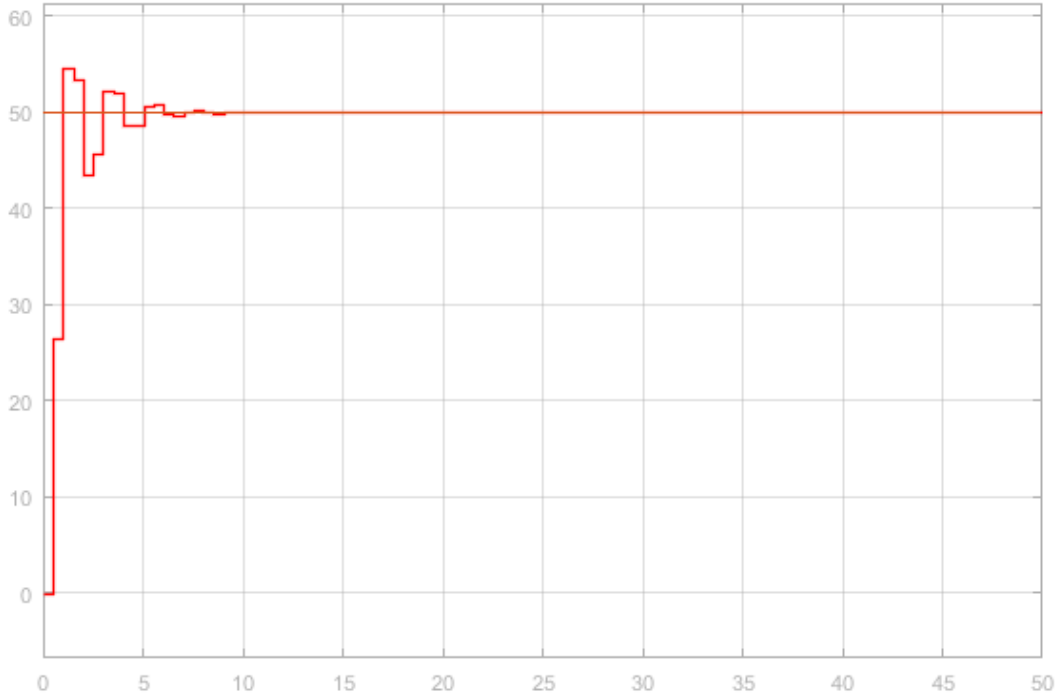
PI results

controller

Kp	1.3
Ki	2.438
Kd	0

response

Rise time	0.5
Settling time	5
Overshoot	9%



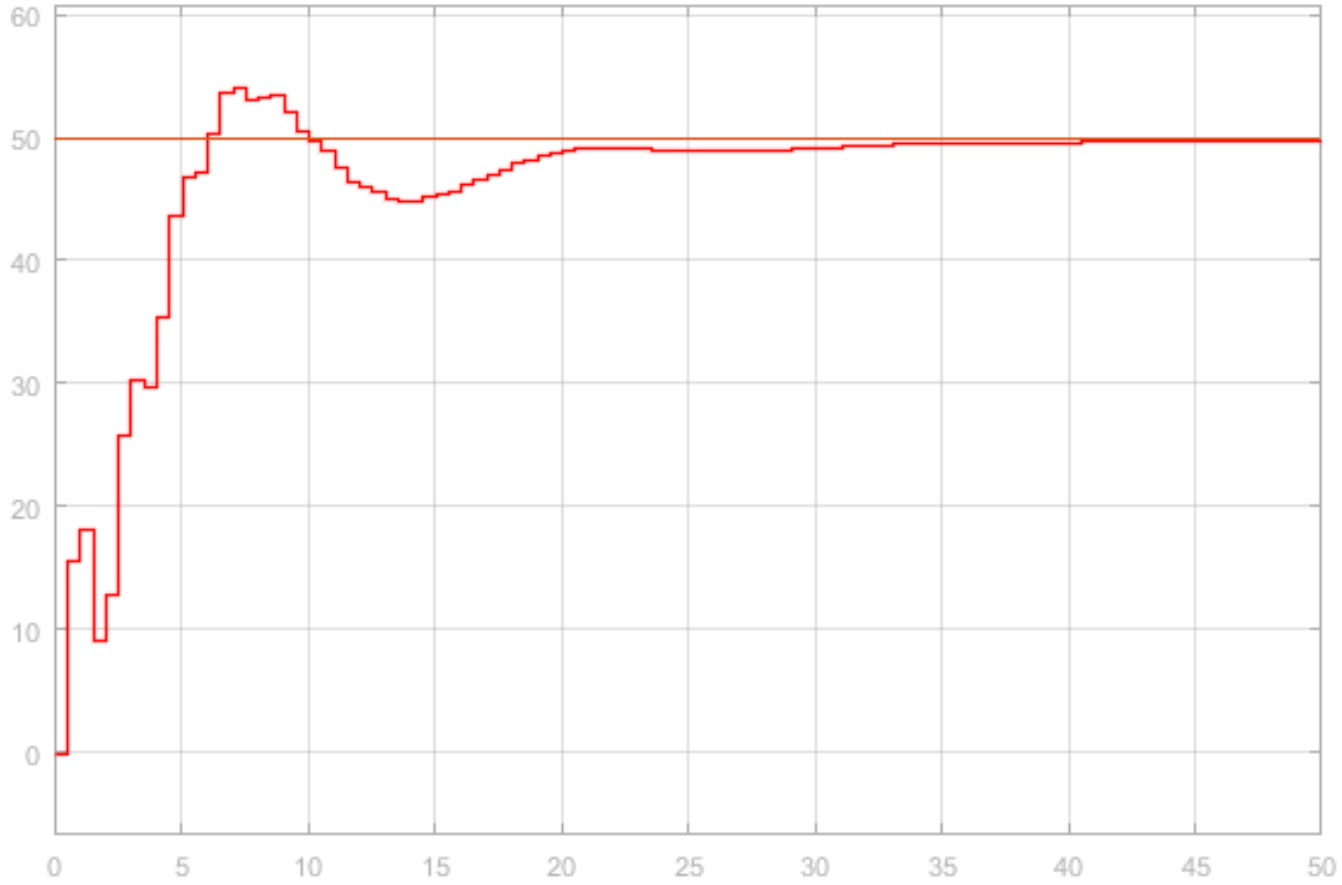
PID Results

controller

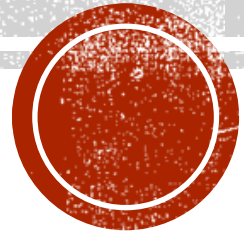
Kp	1.89
Ki	0.255
Kd	4.483

response

Rise time	4.5
Settling time	19.5
Overshoot	7%

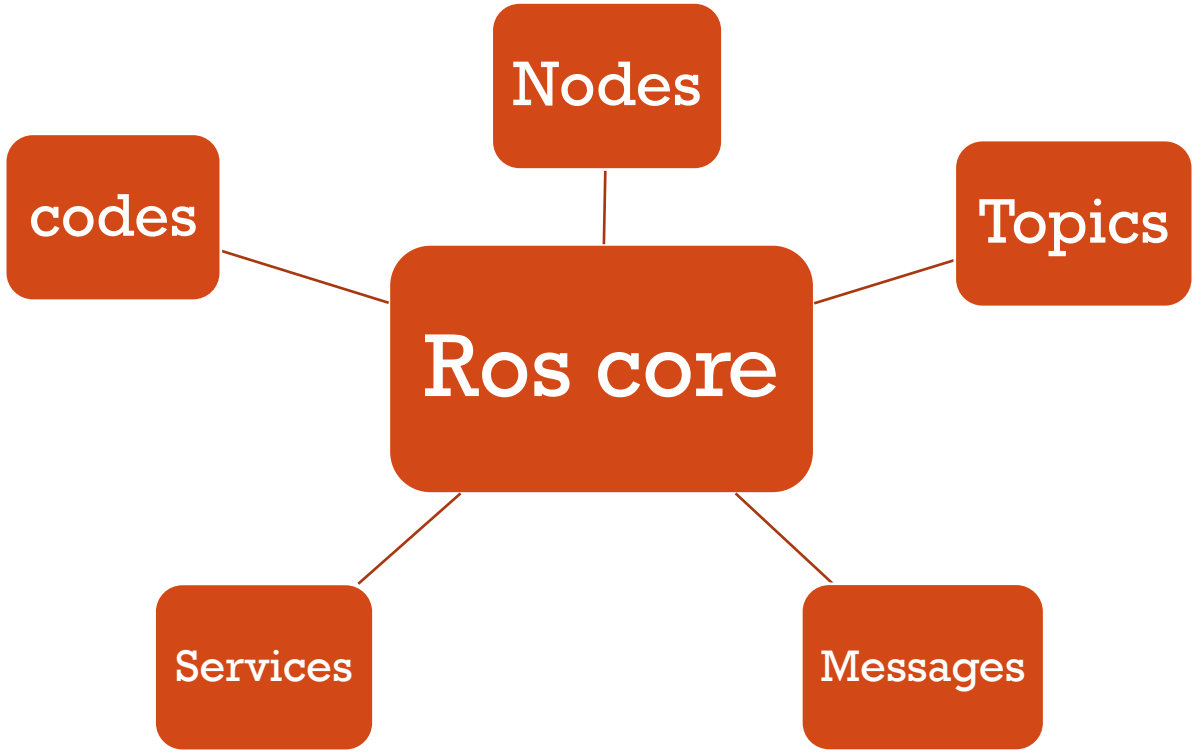


ROBOTIC OPERATING SYSTEM (ROS)



WHAT IS ROS?

- The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms



NODES

- Nodes are processes that perform computation. ROS is designed to be very modular. So, a robot control system usually comprises many nodes. For example, one node controls a laser range-finder
- A ROS node is written with the use of a ROS client library, such as roscpp or rospy.

Messages

- Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, Boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs)
- Example : twist msg (Vector3 linear, vector3 angular)

(x,y,z ; p,q,r)

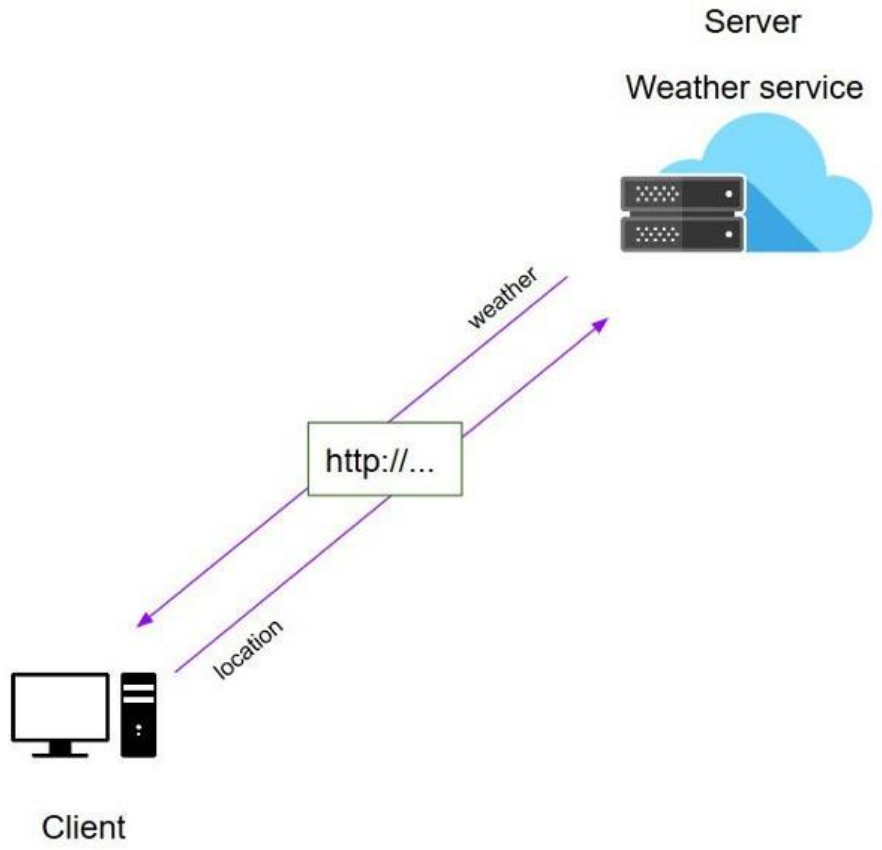
TOPICS

- Messages are routed via a transport system with publish / subscribe semantics

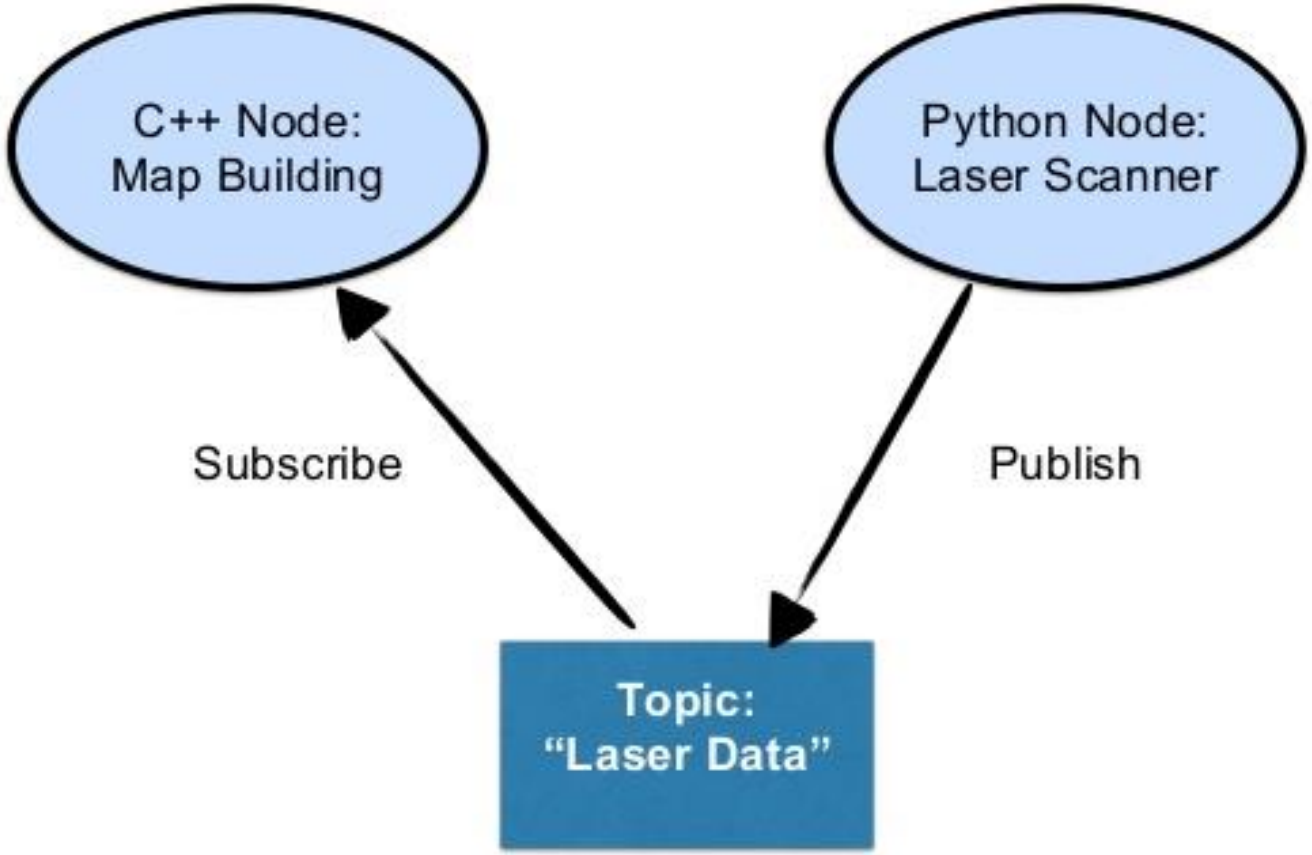
Services

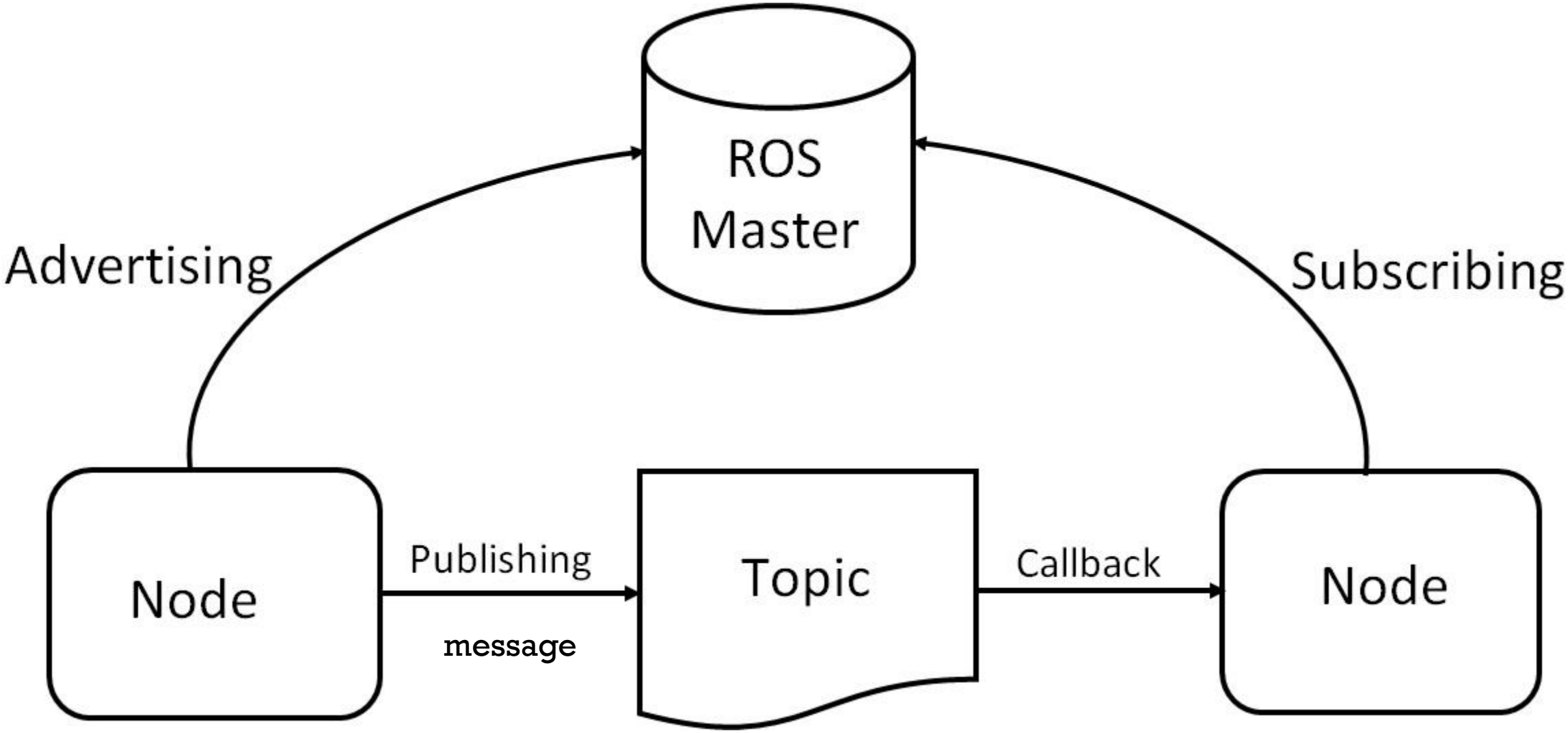
- Is a client/server system

- A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply



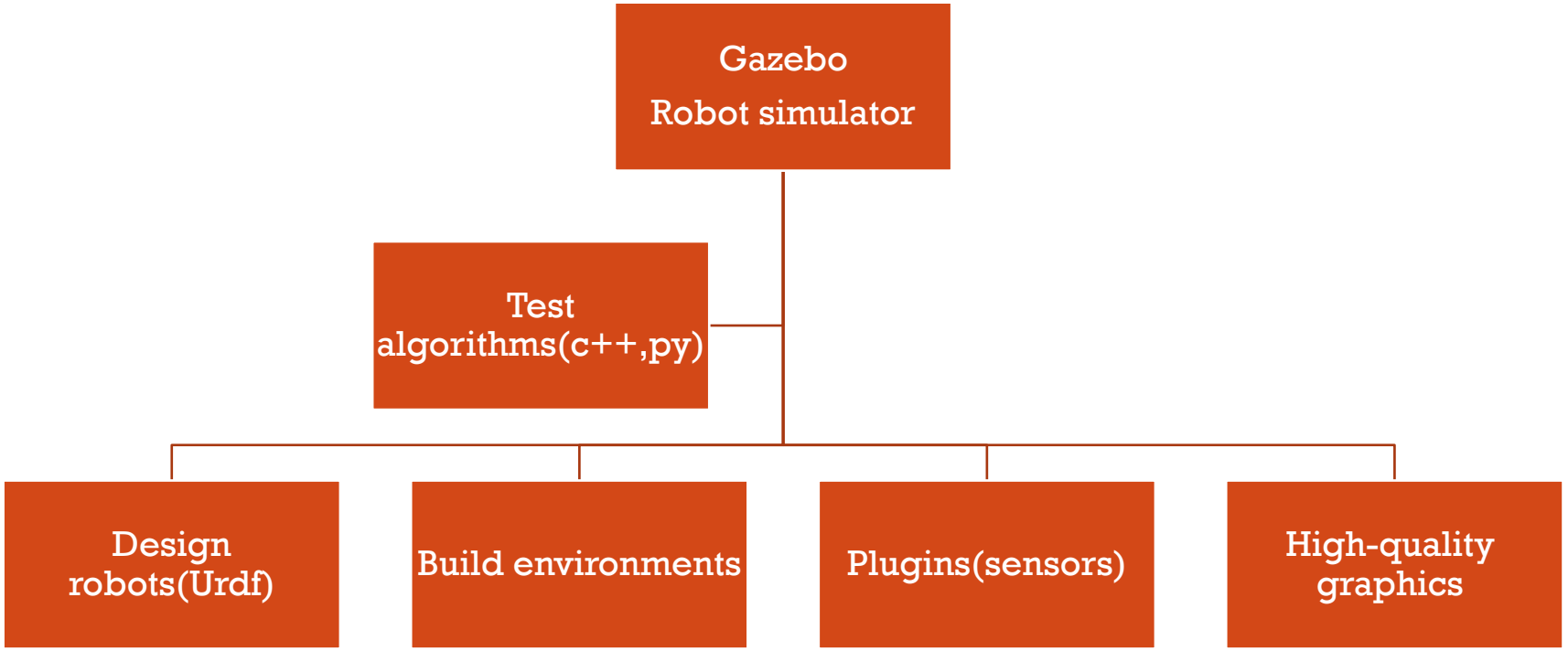
Multiple Languages

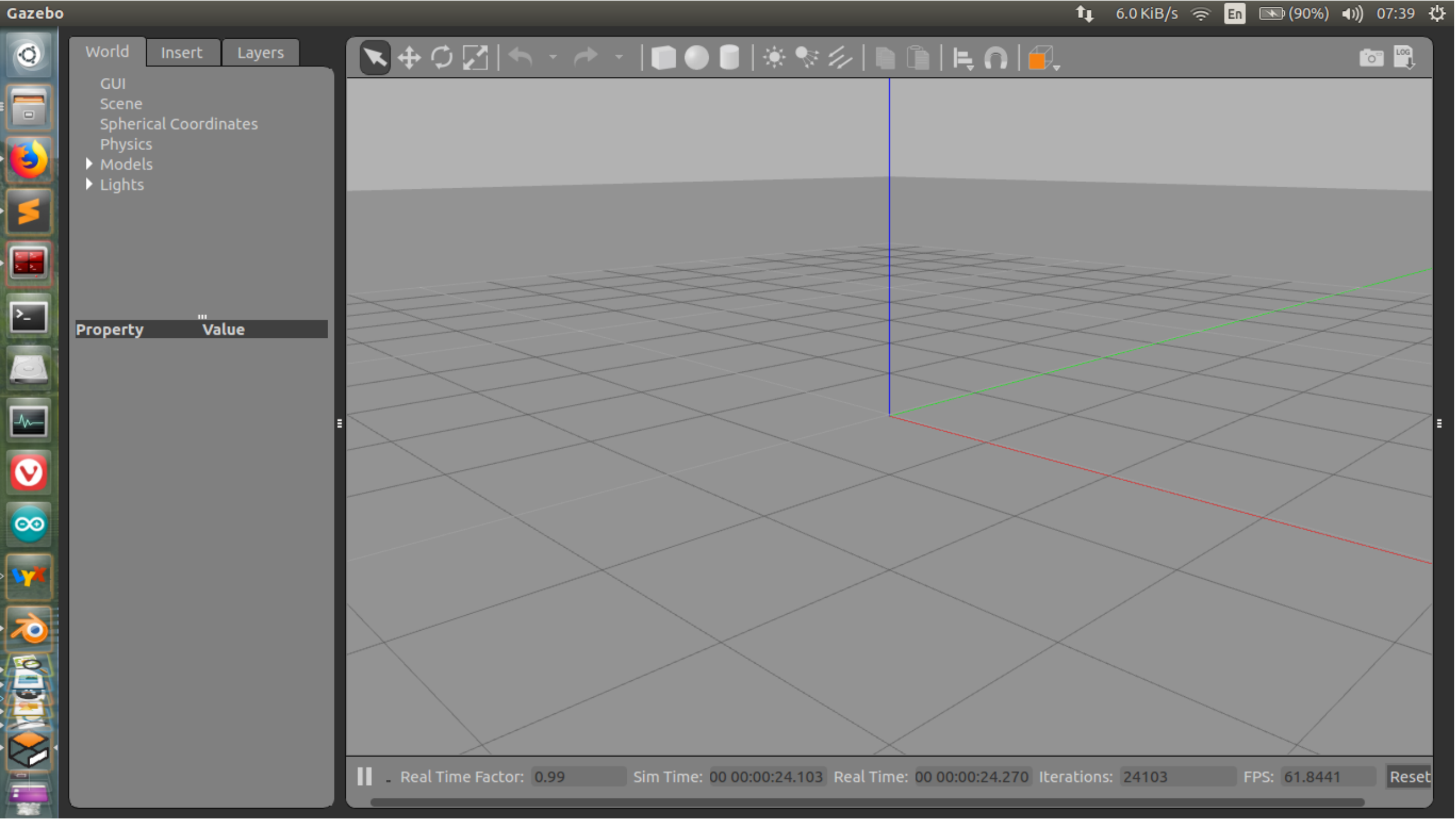



ROS
Master
Advertising
Subscribing
Node
Publishing
message
Topic
Callback
Node

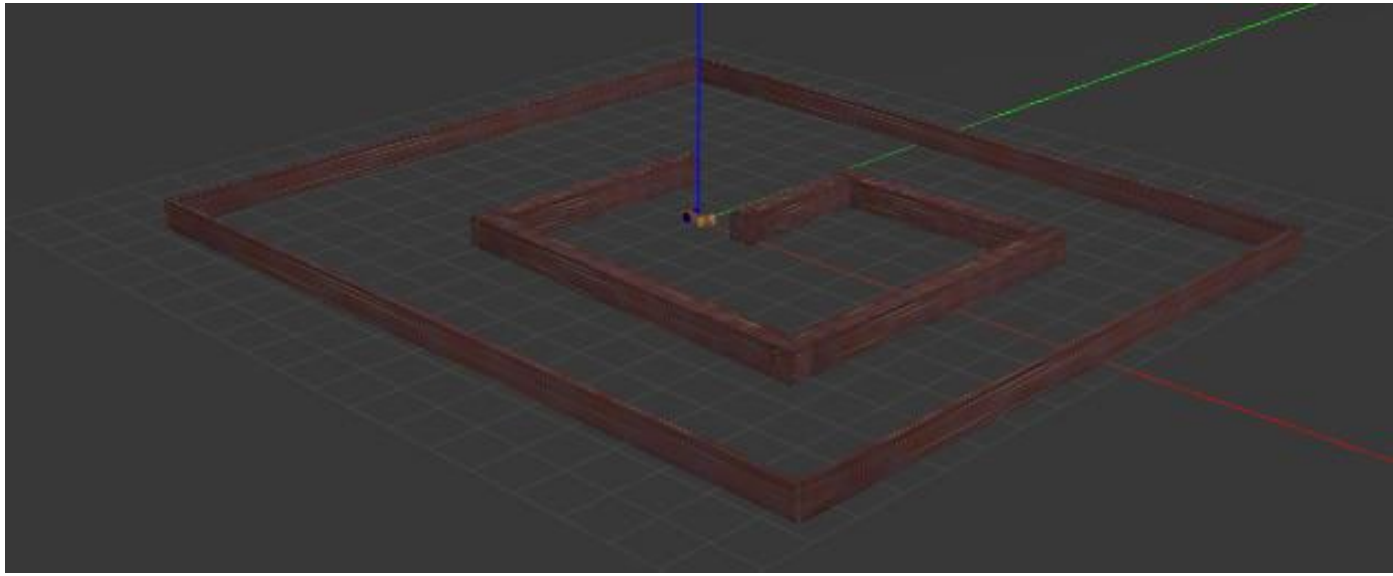
ROS Tools

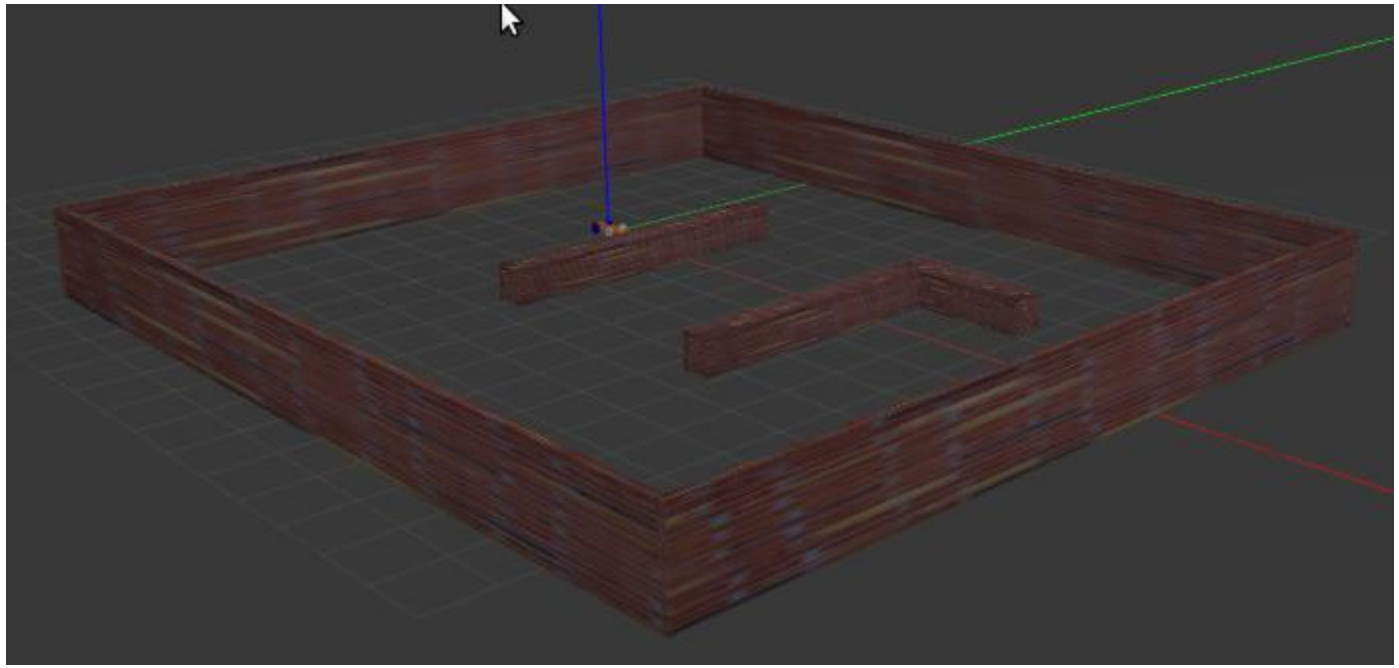
Gazebo	Rviz


Gazebo
Robot simulator
Test algorithms(c++,py)
Design robots(Urdf)
Build environments
Plugins(sensors)
High-quality graphics


Gazebo
6.0 KiB/s
En
(90%)
07:39
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property
Value
Real Time Factor: 0.99
Sim Time: 00 00:00:24.103
Real Time: 00 00:00:24.270
Iterations: 24103
FPS: 61.8441
Reset

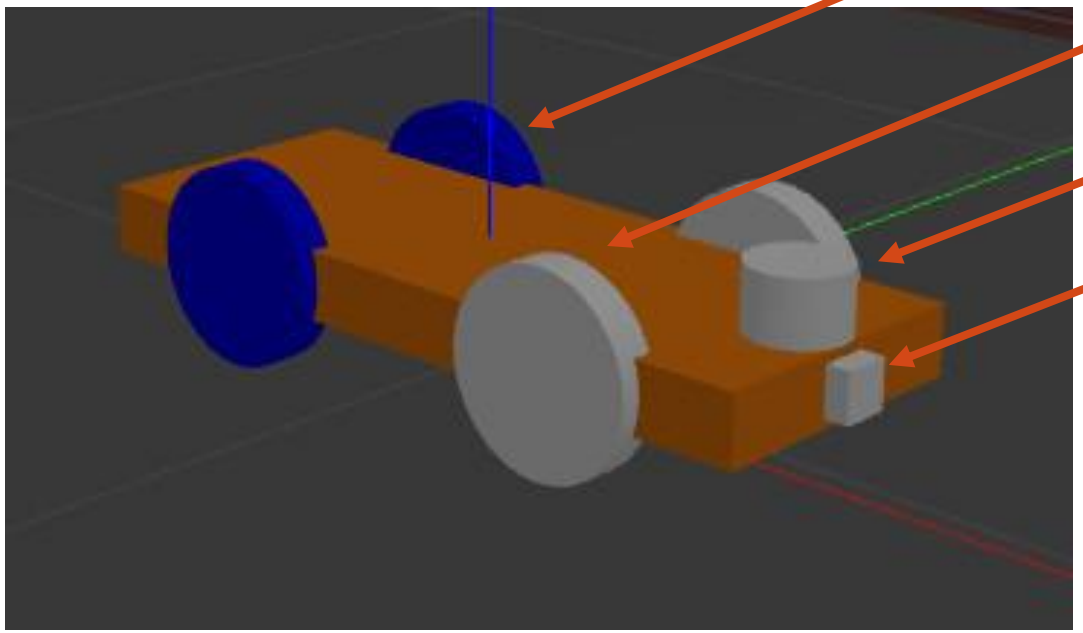
Different environments(worlds)

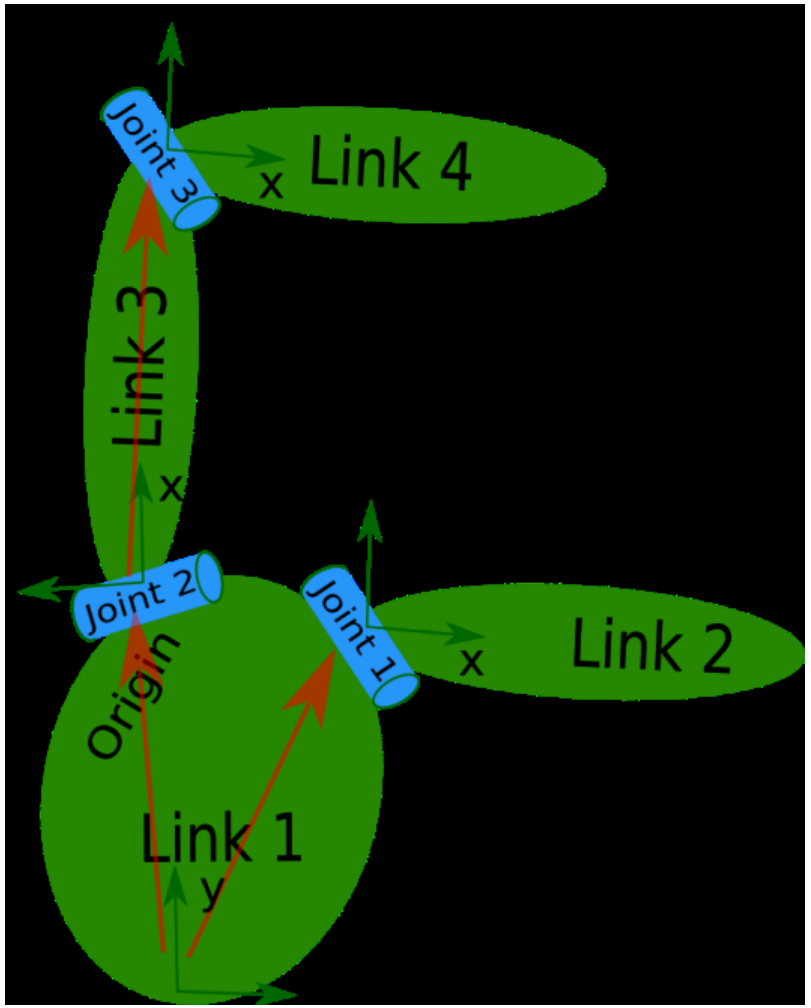




URDF:

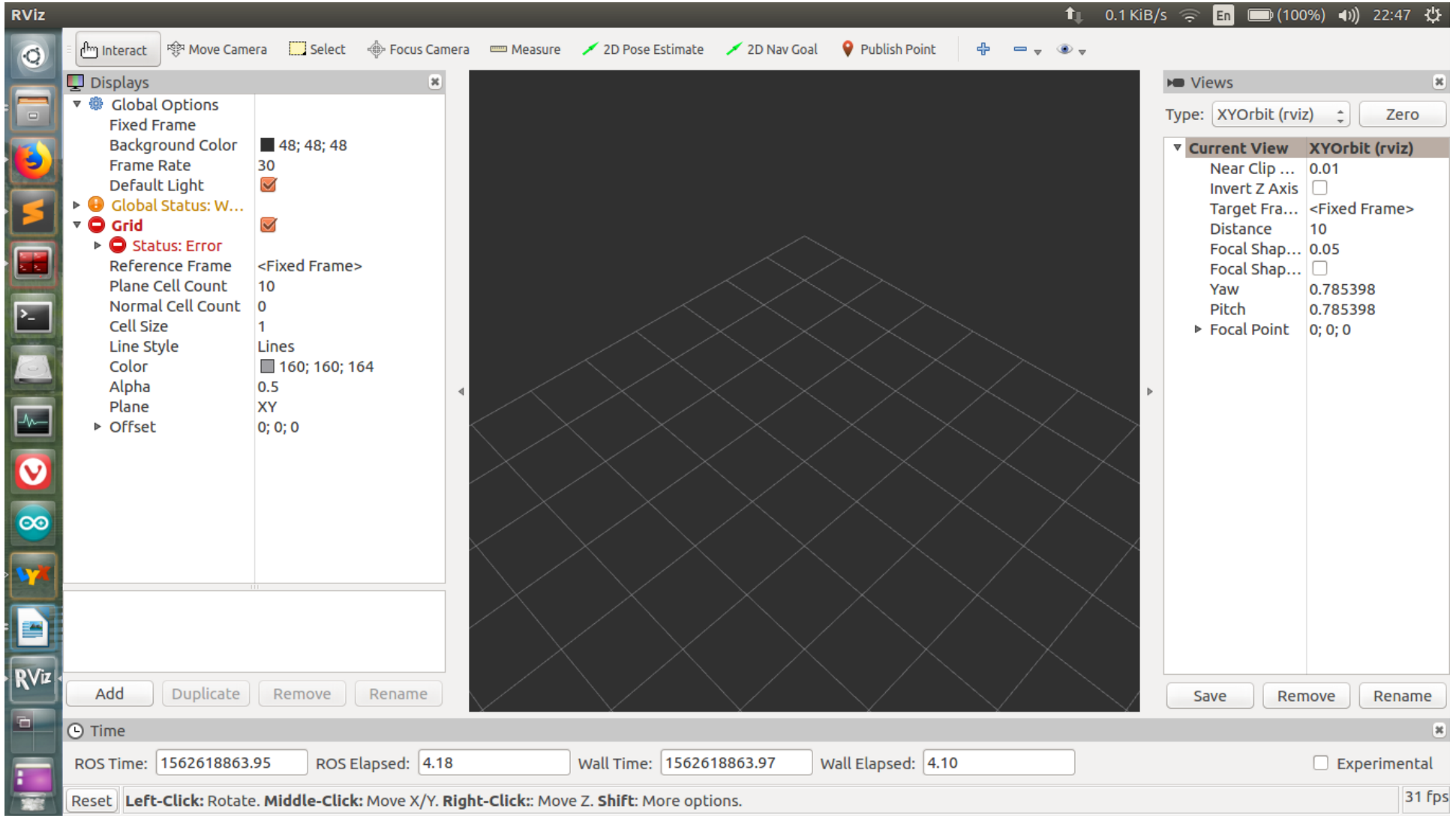
The Unified Robot Description Format (URDF) is an XML specification to describe a robot. URDF is used in ROS to write robot description for use in simulation and visualization software.





RVIZ

- is a ROS graphical interface that allows you to visualize a lot of information, using plugins for many kinds of available topics.



SLAM

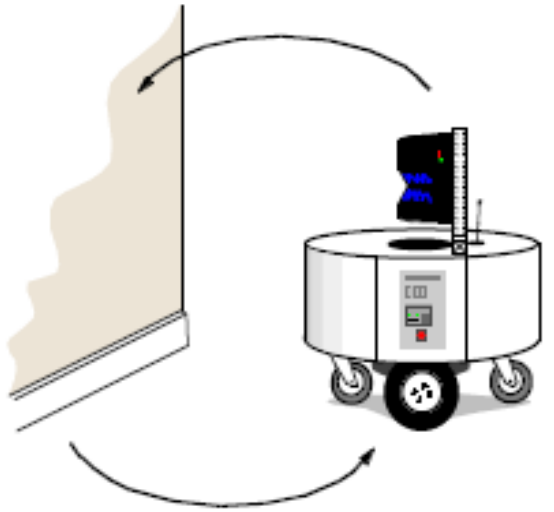
Simultaneous Localization And Mapping

1. WHAT IS SLAM?

- simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.

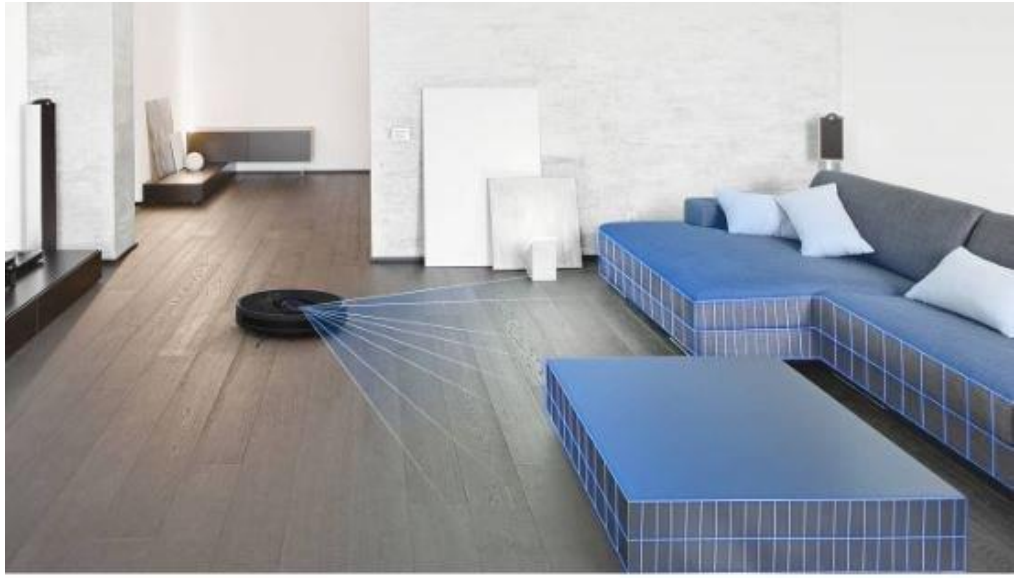
WE CAN SAY THAT SLAM IS A CHICKEN-OR-EGG PROBLEM AS:

- A map is needed for localization and
- A pose estimate is needed for mapping.

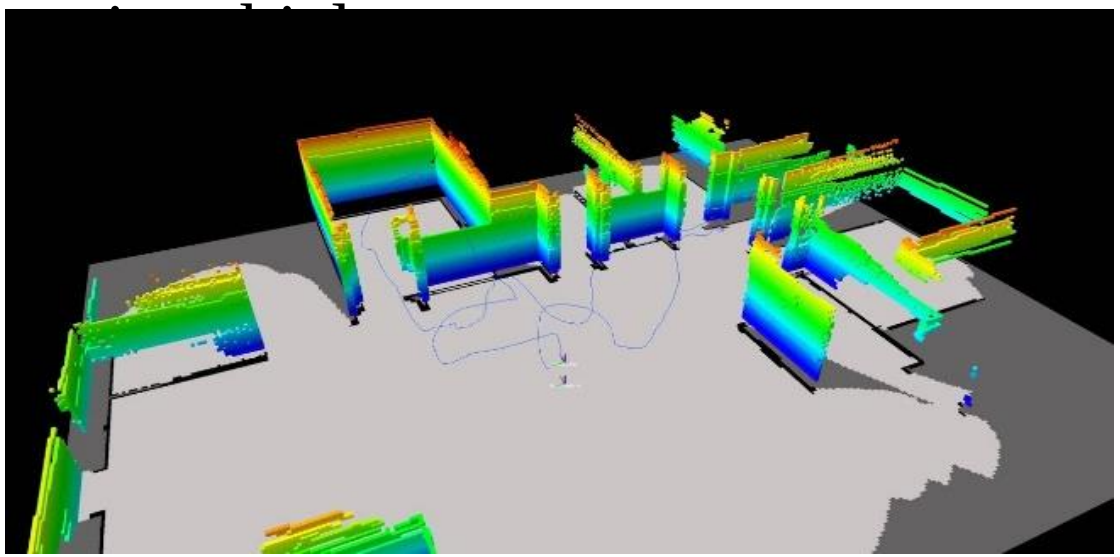


SLAM APPLICATIONS

- vacuum cleaner

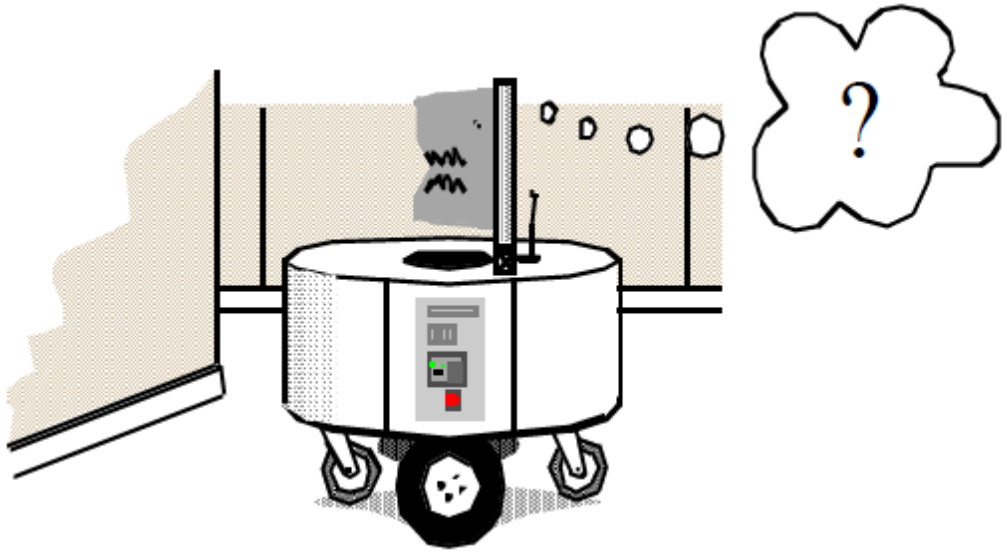


- surveillance with unmanned



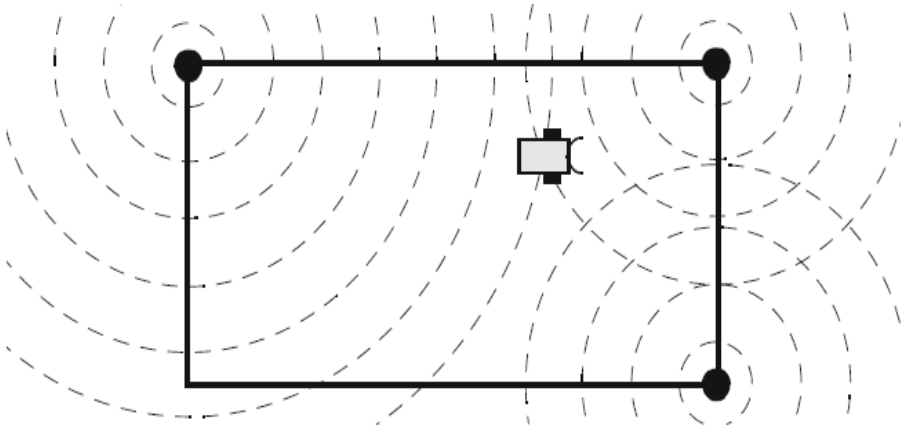
2. LOCALIZATION

- One of the central problems for driving robots is localization. For many application scenarios, we need to know a robot's position and orientation at all times.

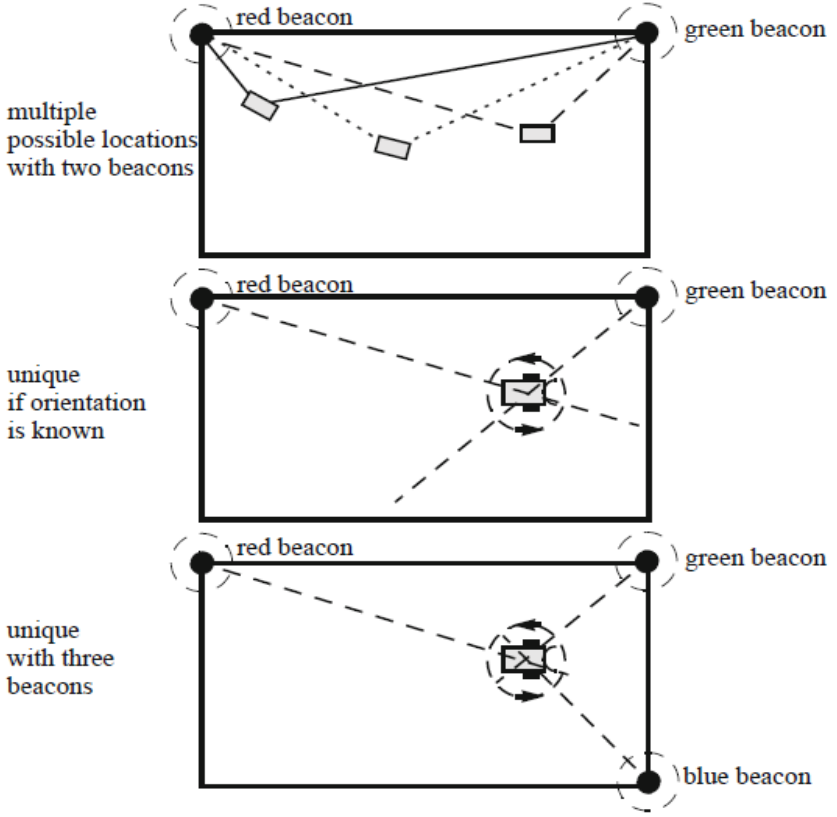


LOCALIZATION USING EXTERNAL SENSORS

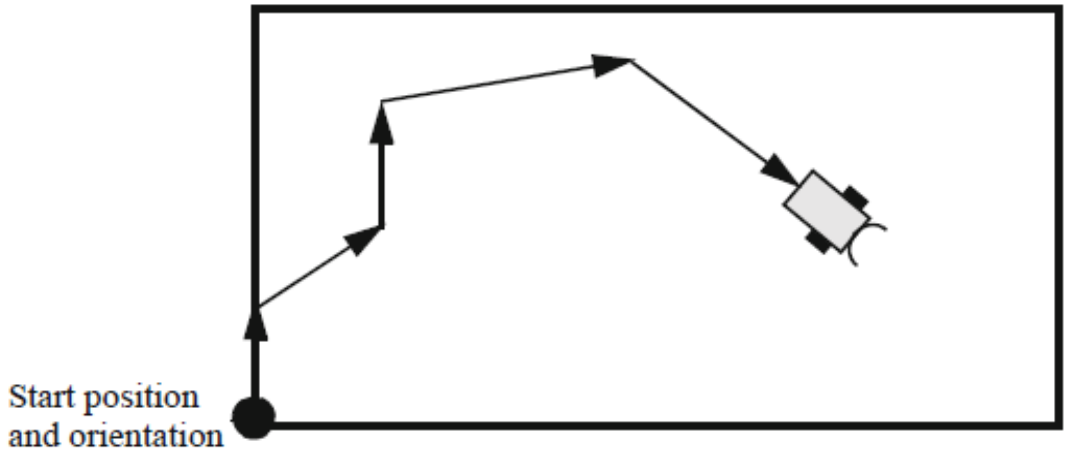
- Using synchronized beacons that are sending out sonar signals at the same regular time intervals, but at different frequencies.



- light emitting homing beacons instead of sonar beacons

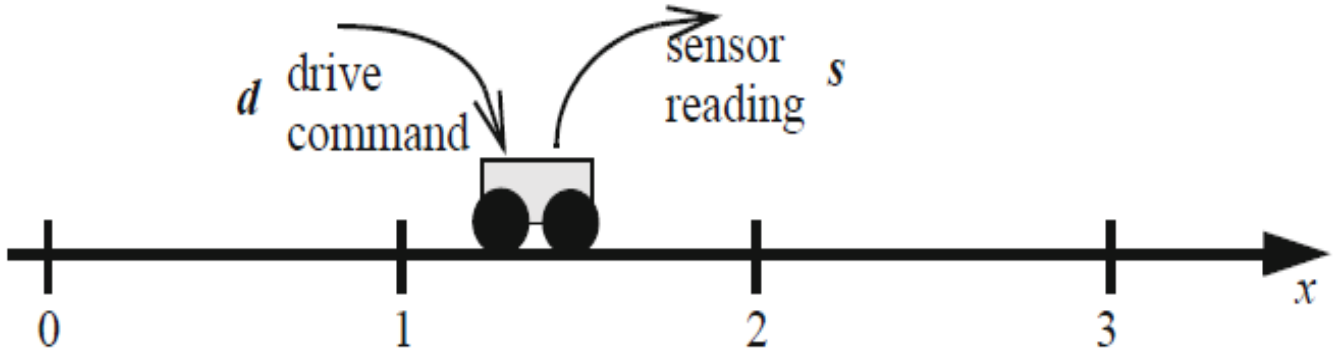


- Dead reckoning
- It is required to know the robot's starting position and orientation. For all subsequent driving actions (for example straight sections or rotations on the spot or curves), the robot's current position is updated as per the feedback provided from the wheel encoders.



3. PROBABILISTIC LOCALIZATION

- All robot motions and sensor measurements are affected by a certain degree of noise. The aim of probabilistic localization is to provide the best possible estimate of the robot's current configurat



EXAMPLE:

- Assume a robot is driving in a straight line along the x axis, starting at the true position x=0. The robot executes driving commands with distance d, where d is an integer, and it receives sensor data from GPS

- The robot's driving accuracy
  - $p(\Delta x = d - 1) = 0.2; p(\Delta x = d) = 0.6; p(\Delta x = d + 1) = 0.2$

- the robot's position sensor accuracy
- $p(x = s - 1) = 0.1; p(x = s) = 0.8; p(x = s + 1) = 0.1$

- Assuming the robot has executed a driving command with $d=2$ and after completion of this command, its local sensor reports its position as $s=2$. The probabilities for its actual position x are as follows, with n as normalization factor:
  - $p(x=1) = n \cdot p(s=2 \mid x=1) \cdot p(x=1 \mid d=2, x'=0) \cdot p(x'=0)$
    $= n \cdot 0.1 \cdot 0.2 \cdot 1 = 0.02n$
  - $p(x=2) = n \cdot p(s=2 \mid x=2) \cdot p(x=2 \mid d=2, x'=0) \cdot p(x'=0)$
    $= n \cdot 0.8 \cdot 0.6 \cdot 1 = 0.48n$
  - $p(x=3) = n \cdot p(s=2 \mid x=3) \cdot p(x=3 \mid d=2, x'=0) \cdot p(x'=0)$
    $= n \cdot 0.1 \cdot 0.2 \cdot 1 = 0.02n$

- $0.02n + 0.48n + 0.02n = 1$

$$n = 1.92$$

- Now, we can calculate the probabilities for the three positions, which reflect the robot's belief:
  - $p(x=1) = 0.04$;
  - $p(x=2) = 0.92$;
  - $p(x=3) = 0.04$

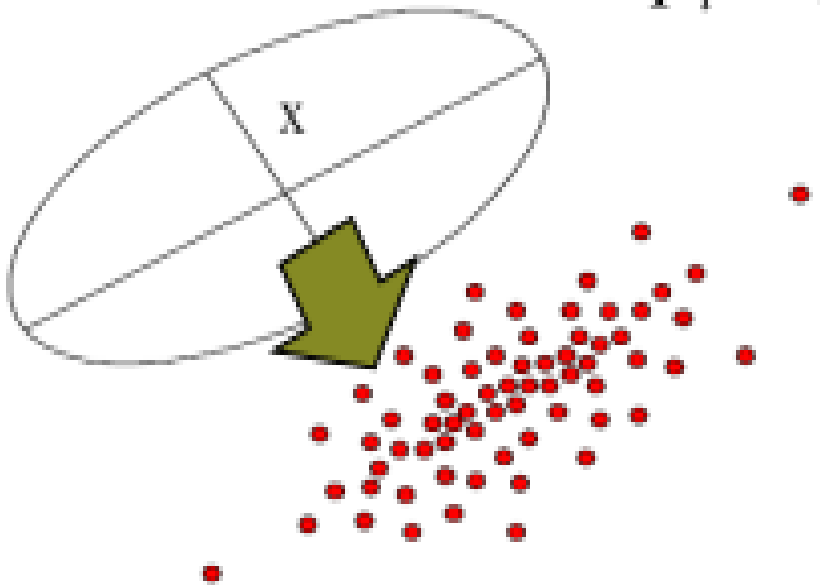
THE PROBLEM WITH THIS APPROACH

- The biggest problem with this approach is that the configuration space must be discrete. That is, the robot's position can only be represented discretely.
  - To overcome this problem we use SLAM using particle filter.

OVERVIEW OF PARTICLE FILTER

- allow the use of non-discrete configuration spaces. The key idea in particle filters is to represent the robot's belief as a set of N particles, collectively known as M. Each particle consists of a robot configuratio

$$p_i = \{y_{t_i}, w_i\}$$



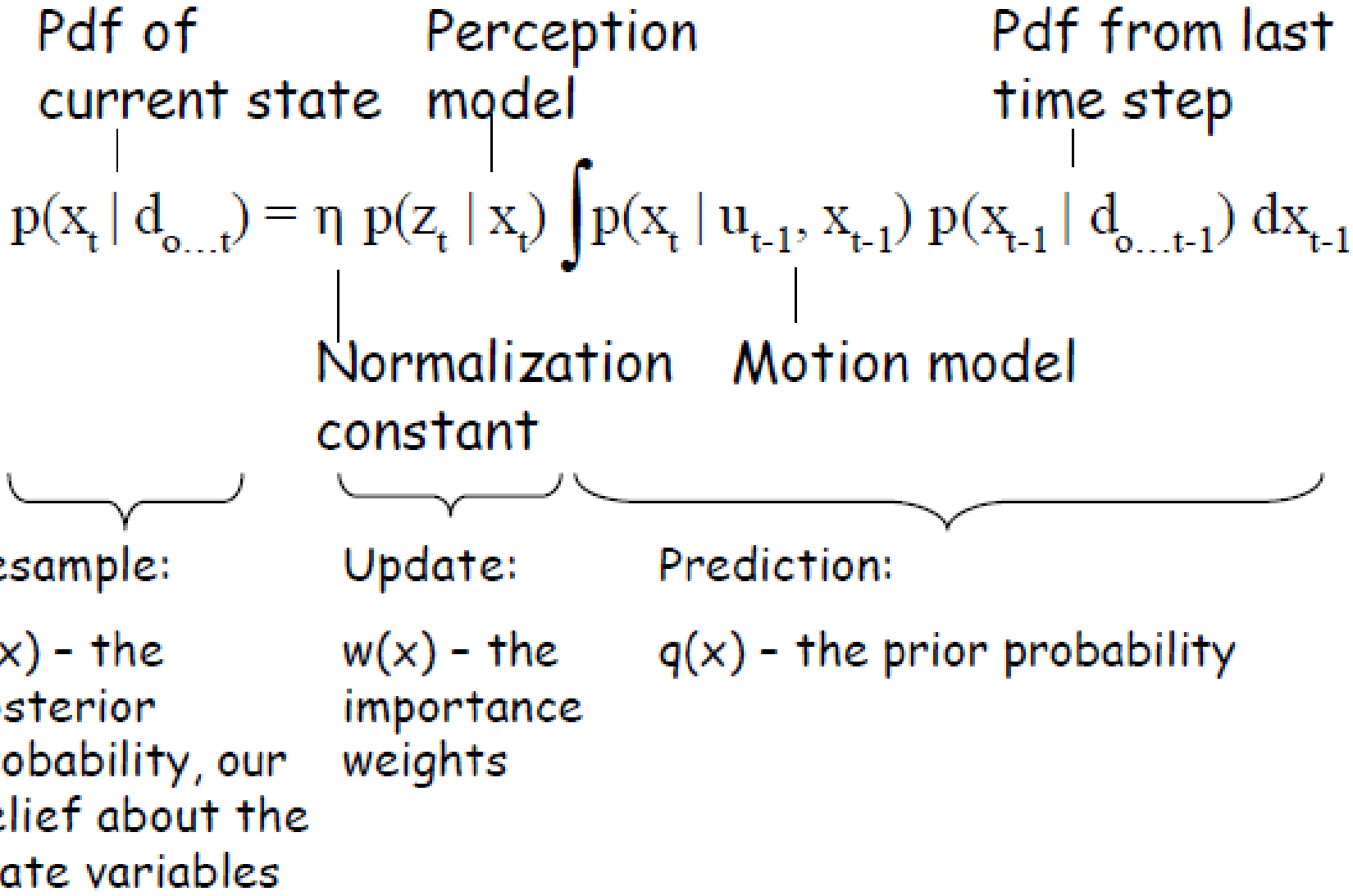
- Represents pdf as a set of samples (particles)
- Each particle contains one set of values for the state variables
- Good for non-Gaussian, multi-modal pdfs
- Find an approximate solution using a complex model (arbitary pdf (probability density function)) rather than an exact solution using a simplified model (Gaussians)

THE ALGORITHM

- To start: Sample from initial pdf, p(x0)
  - For localization, no idea where robot is → throw particles everywhere!
- For each time step, loop with three phases:

1) Prediction
2) Update
3) Resample

CALCULATION OF BELIEF FOR ROBOT LO

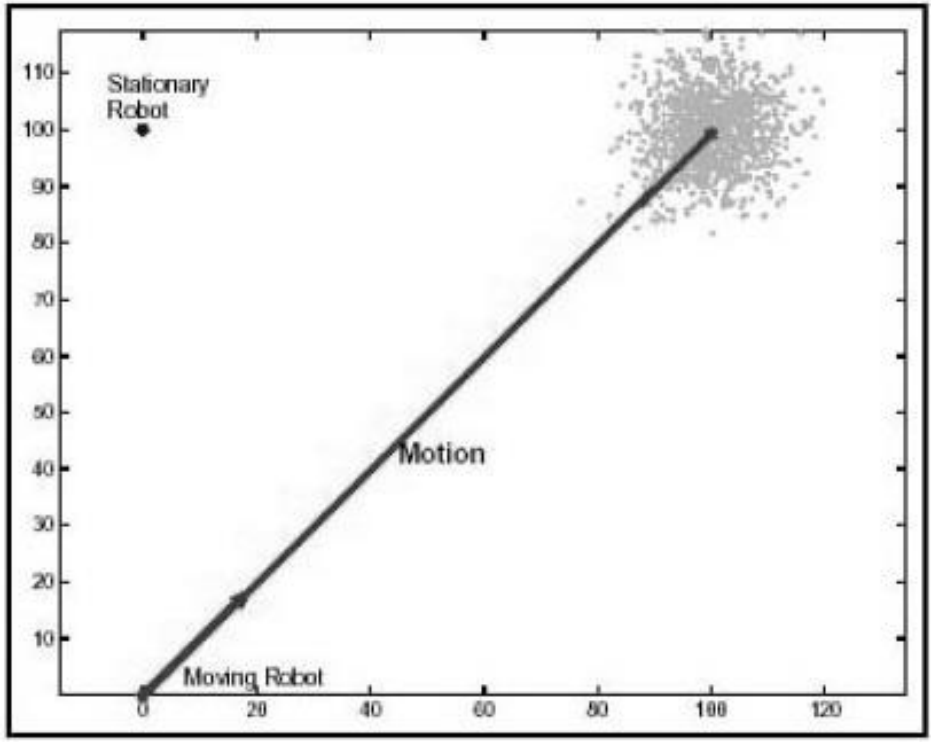


A) PREDICTION

- For each particle, sample and add random, noisy values from action model
- Resulting proposal distribution (q(x))

  approximates

$$\int p(x_t|x_{t-1}, u_{t-1})p(x_{t-1}|d_{p..t-1})dx_{t-}$$



B) UPDATE

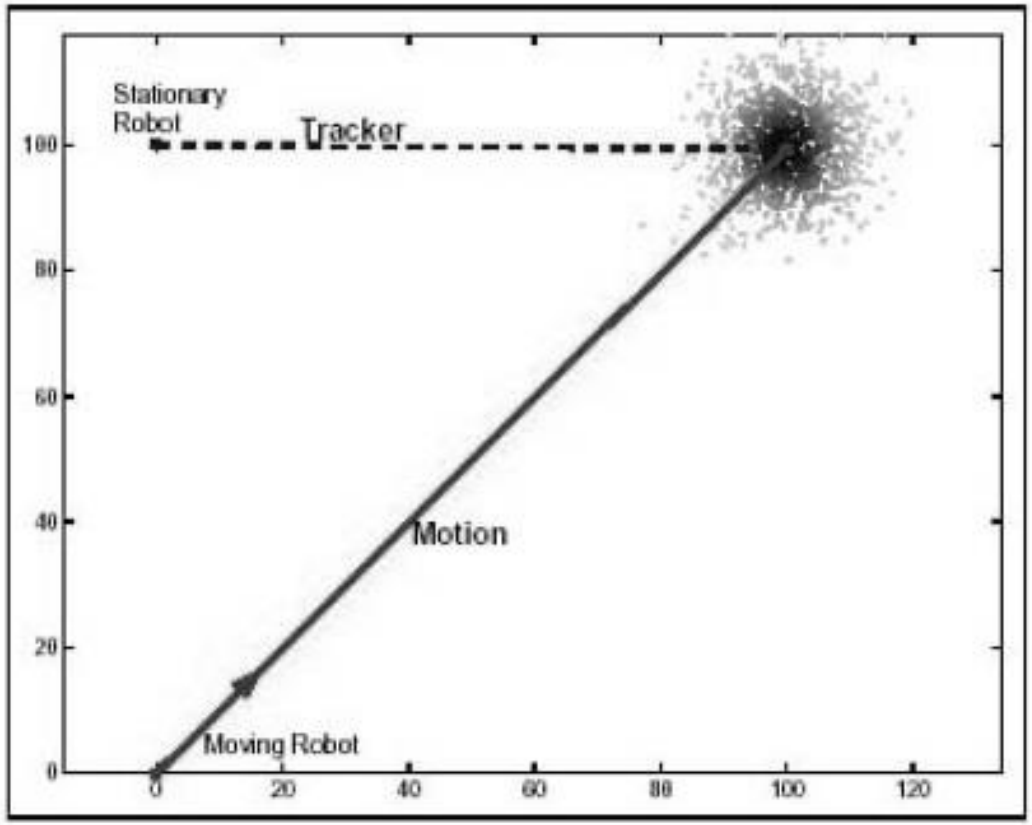
- each particle's weight is the likelihood of getting th
  hypothesis
- The weight associated with

  each particle is

  $w(x) = p(x)/q(x) = p(z_t \mid x_t)$,

  normalized so that all the weights

  sum to 1



C) RESAMPLE

- New set of particles are chosen such that each particle survives in proportion to its weight

Resulting distribution is p(x):

$$\underbrace{p(x_t \mid d_{o\ldots t})}_{p(x)} = \underbrace{\eta\, p(z_t \mid x_t)}_{w(x)} \underbrace{\int p(x_t \mid u_{t-1}, x_{t-1})\, p(x_{t-1} \mid d_{o\ldots t-1})\, dx_{t-1}}_{q(x)}$$

p(x) – the posterior probability

w(x) – the importance weights

q(x) – the prior probability

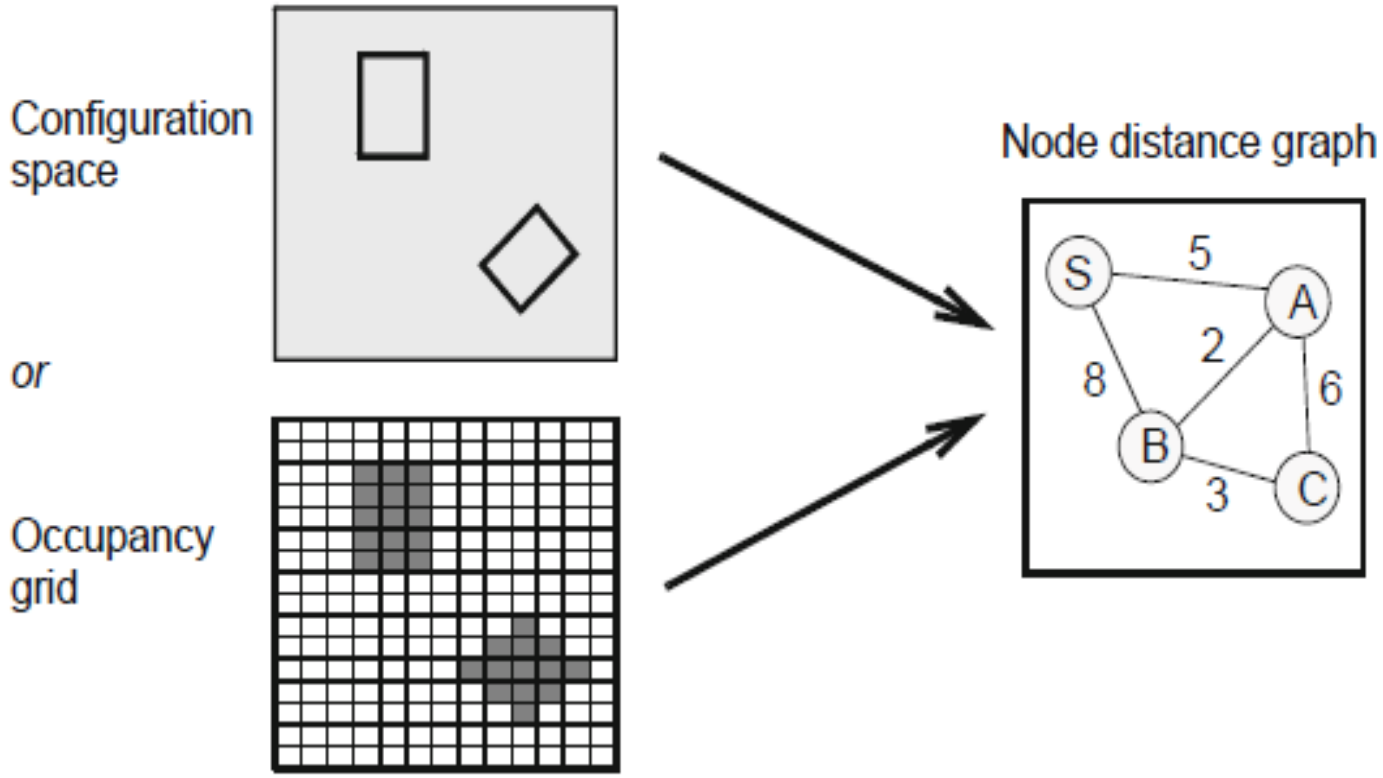
4. ENVIRONMENT REPRESENTATION

a) 2D occupancy grid using Laser sensor (LIDAR)

b) 3D mapping using RGB-D sensor (Xbox360 Kinect)

A) 2D OCCUPANCY GRID USING LASER SENSOR (LIDAR)

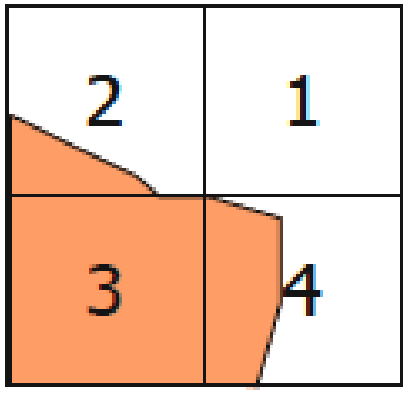
- the environment is specified at a certain resolution with individual pixels either representing free space (white pixels) or an obstacle (black pixels).

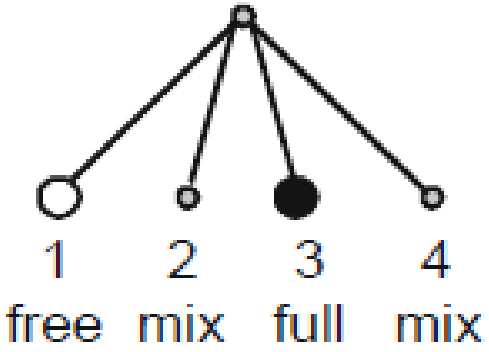


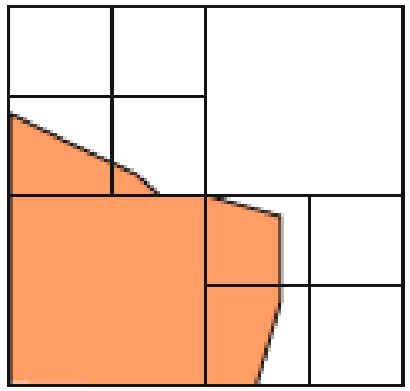
THIS APPROACH HAS A NUMBER OF PROBLEMS

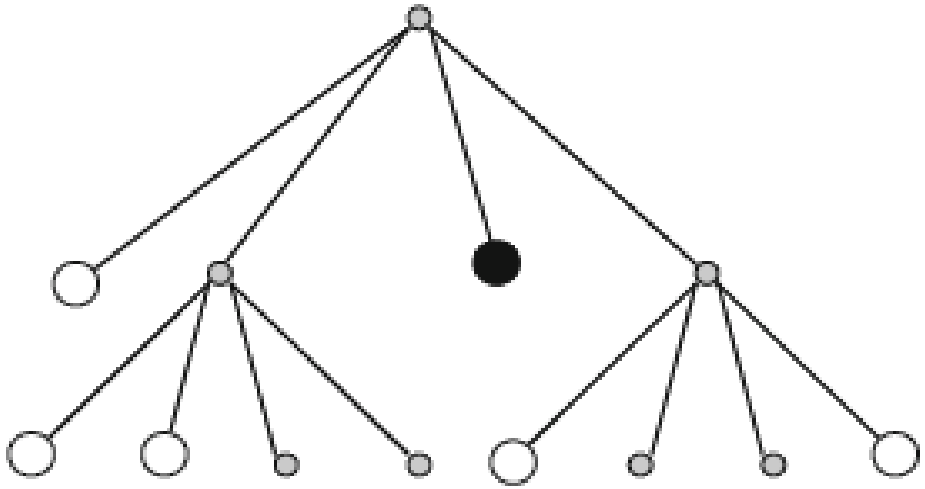
- The number of nodes in the resulting distance graph will be huge
- As neighboring pixels have been transformed into neighboring graph nodes and therefore only support turning angles that are multiples of ±45º (eight nearest neighbors) of multiples of ±90º (four nearest neighbors).

USING A QUADTREE WILL IMPROVE THIS SITUATION ON BOTH COUNTS



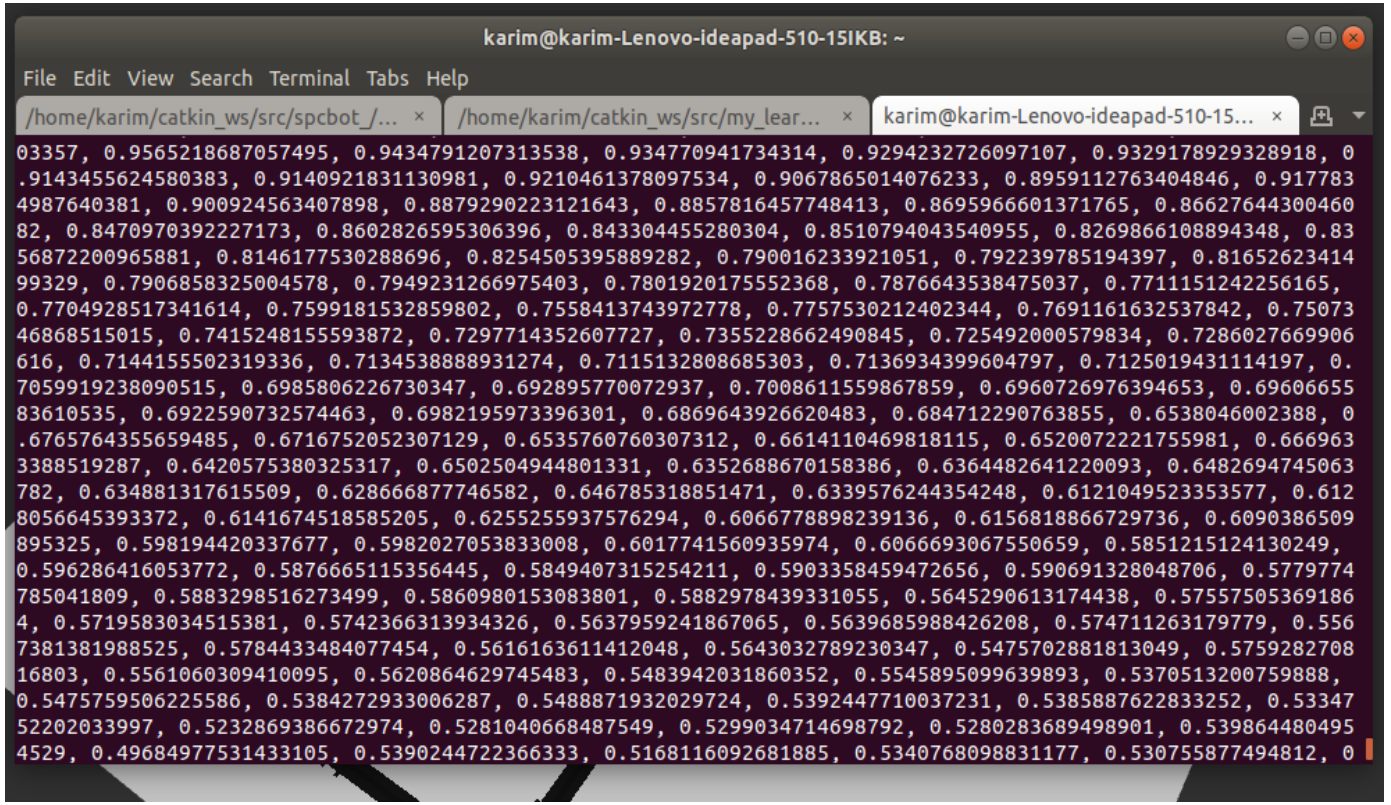




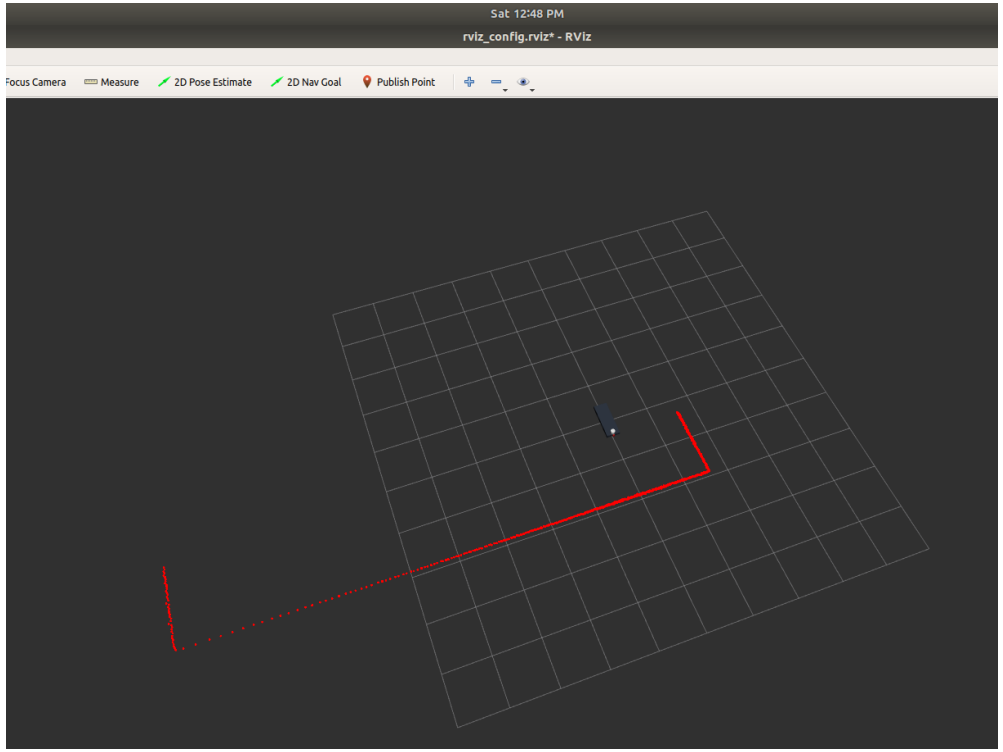


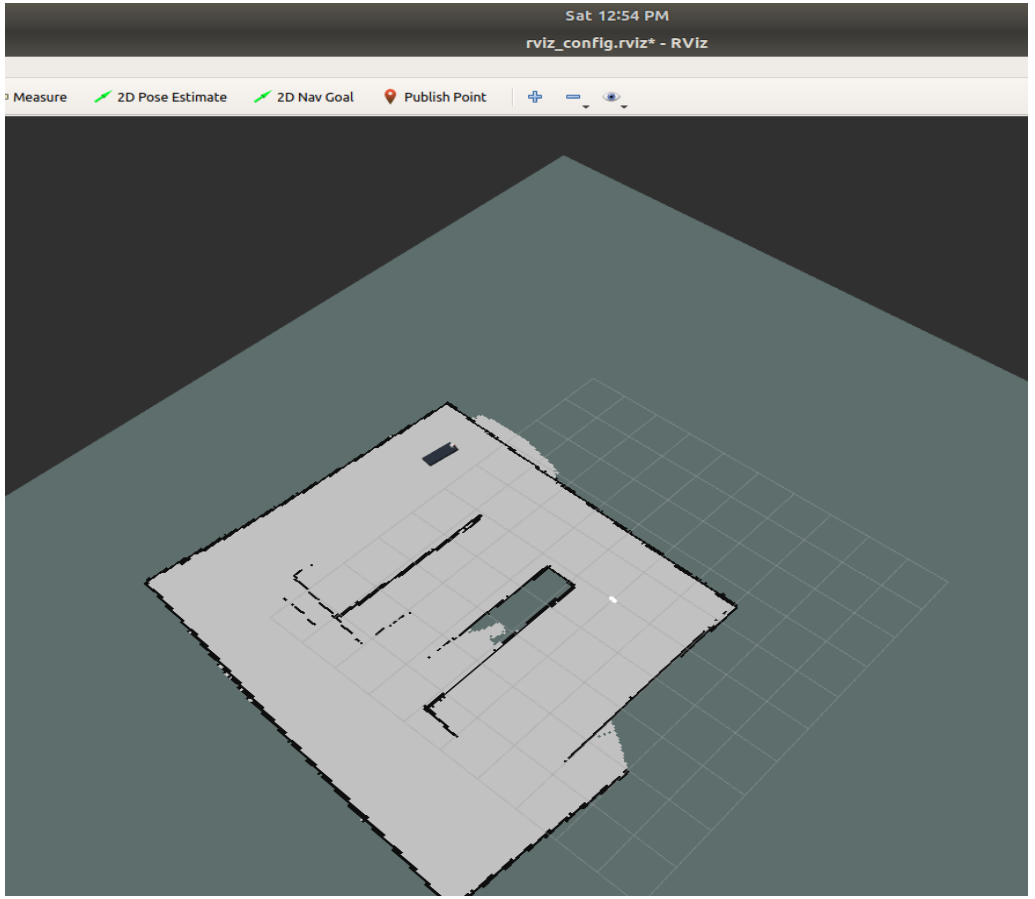
2D GRID MAP USING LIDAR

- First we use YDLIDAR ROS package to get readings from the LIDAR (laser_scan topic)



- Then, we use gmapping ROS package which uses readings from the LIDAR by subscribing to laser_scan topic which is now published by the YDLIDAR ROS package



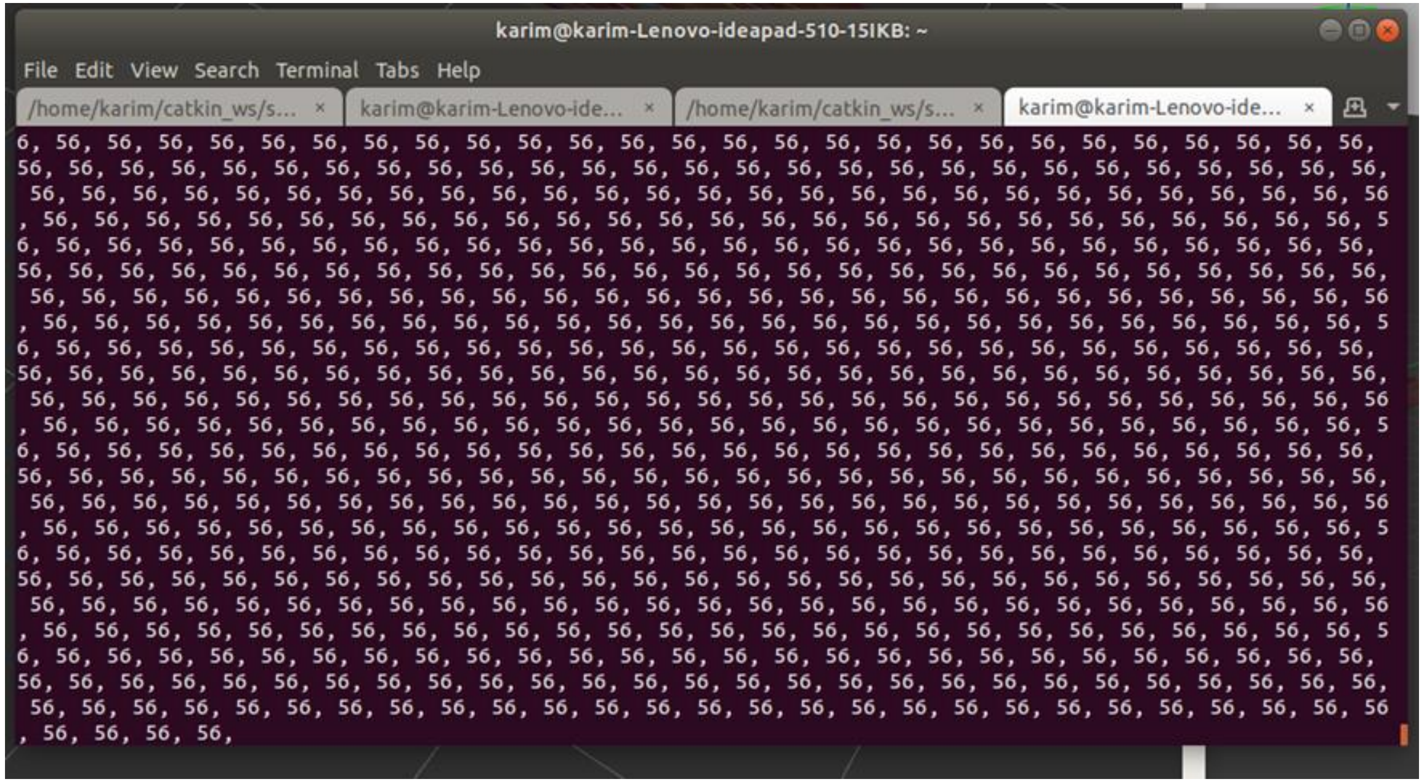


ILLUSTRATING VIDEO (BUILDING 2D MAP BY LIDAR)

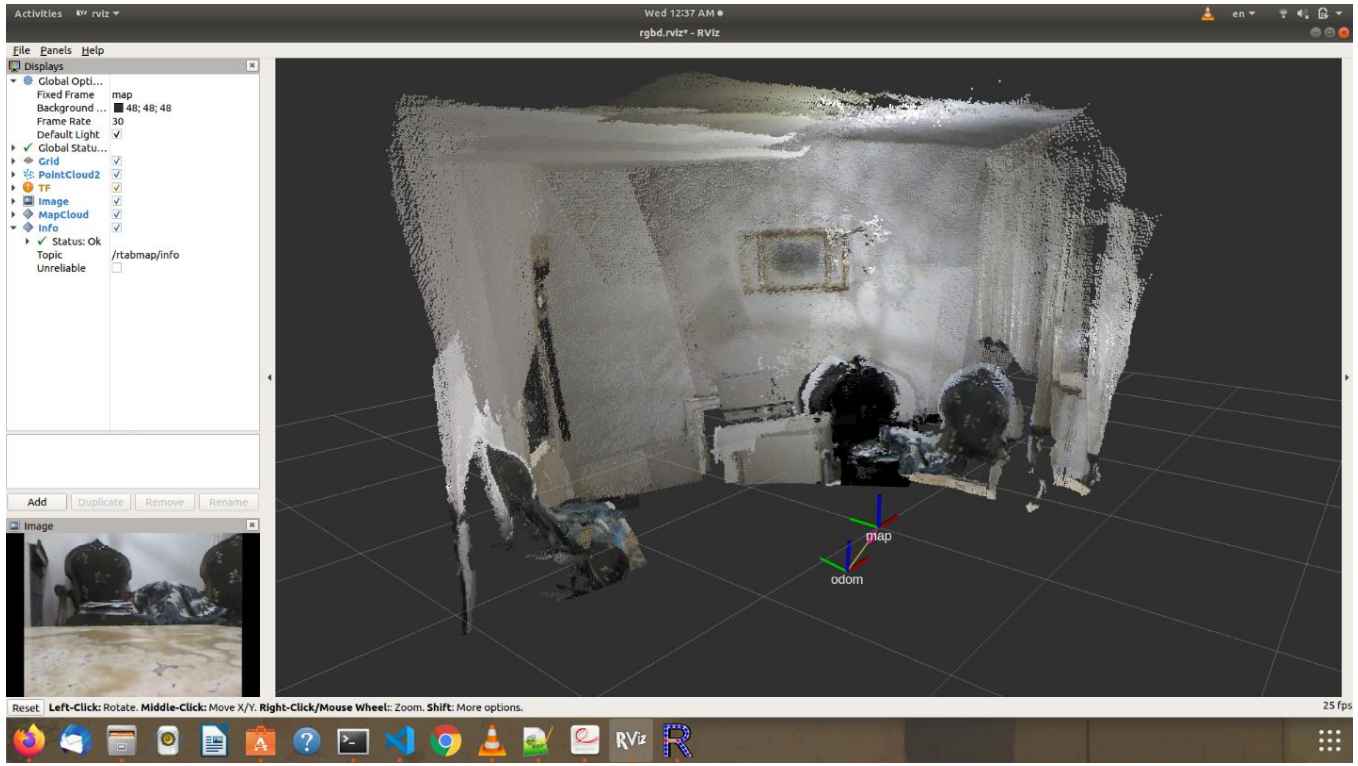
- VIDEO 1

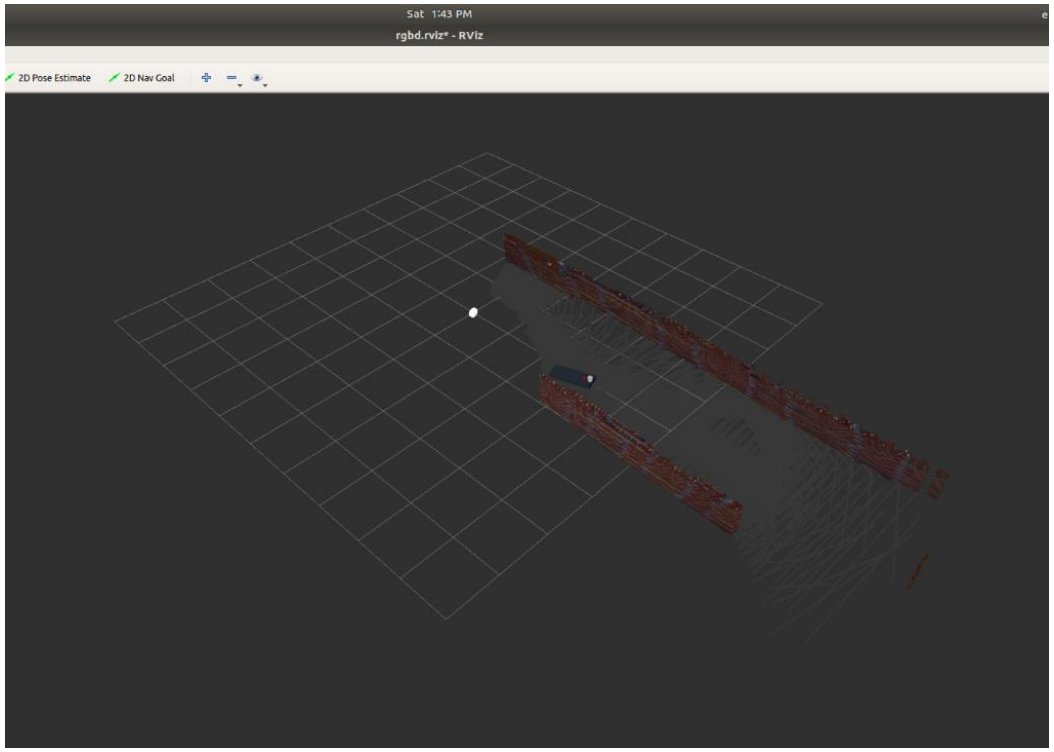
3D MAPPING USING RGB-D SENSOR (XBOX360 KINECT)

- First we use OpenNI ROS package to get readings from the Kinect camera



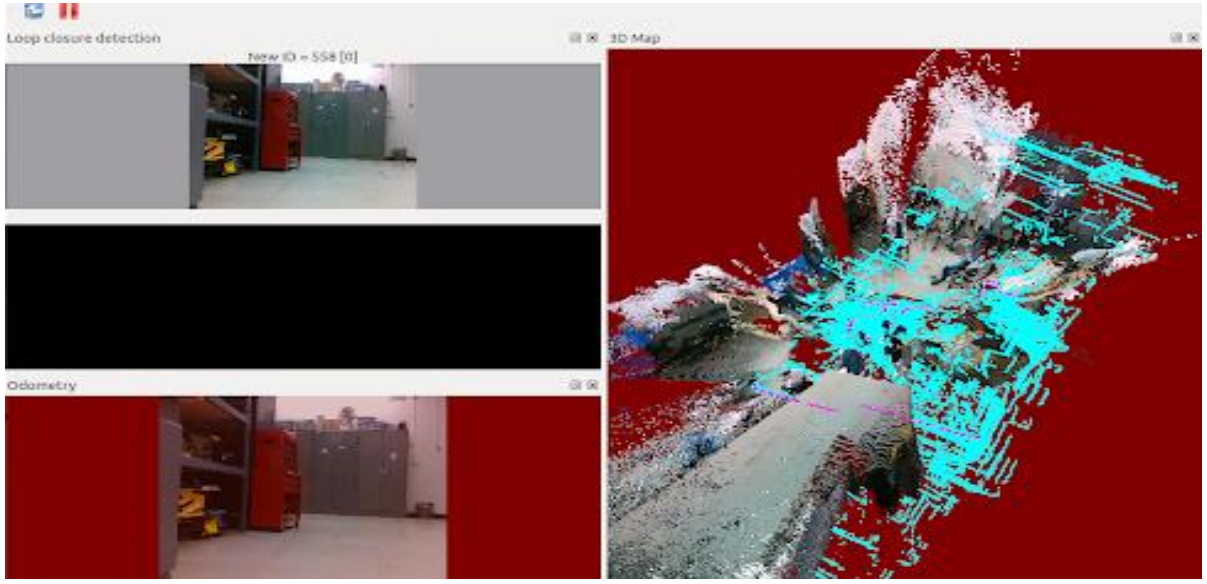
- The second package used is RTABmap_ros (Real-Time Appearance-Based Mapping). It uses this reading that we get from the Kinect camera and perform RGB-D SLAM on it



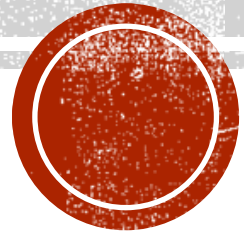


5. COMPARISON BETWEEN 2D AND 3D MAPS

- 2D LIDAR sensor is more accurate than Kinect sensor, covers more distance and covers an angle of 360^{o} of the surroundings. The Kinect sensor covers only about 57^{o}.
- Kinect camera is better for obstacle avoidance as it detects the height of the obstacle
- The Kinect camera has much noises and larger errors than LIDAR sensor.
- Kinect camera needs to move much slower to avoid loop closure failure on creating the map.



MOTION PLANNING



MOTION PLANNING

Map independent

- Bug 0
- Bug 1
- Bug 2

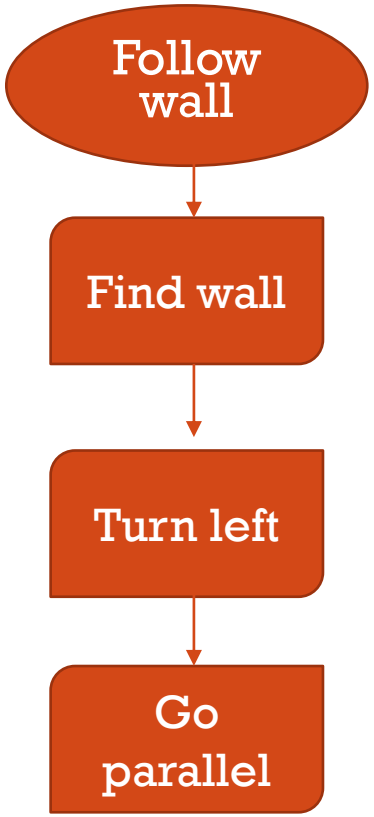
Map dependent

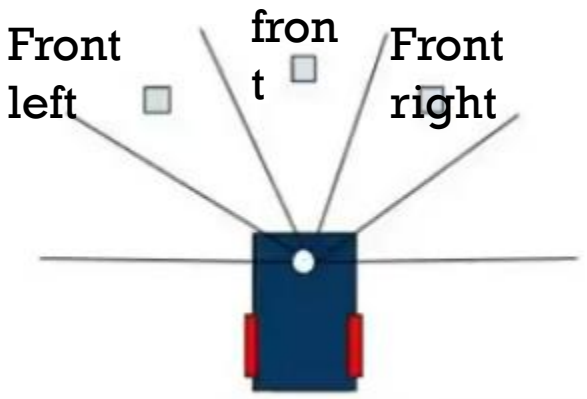
- A star

BASIC ALGORITHMS NEEDED FOR BUGS

- **Go To point**
  - Givens : start point and goal point

* FOLLOW WALL



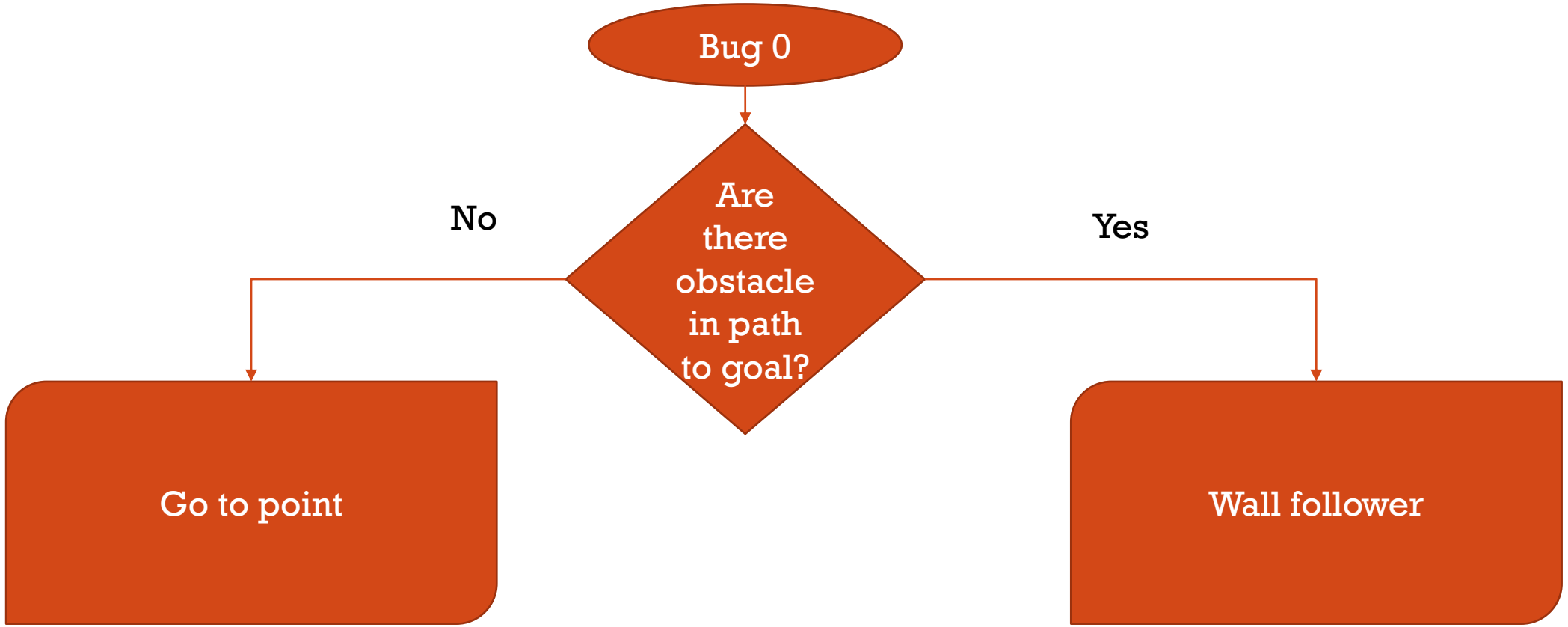


Laser scanner

Case	Front left	Front	Front right	Task
1				Find wall
2		X		Turn left
3			X	Go parallel
4	X			Find wall
5		X	X	Turn left
6	X	X		Turn left
7	X	X	X	Turn left
8	X		X	Find wall

Tasks for different cases

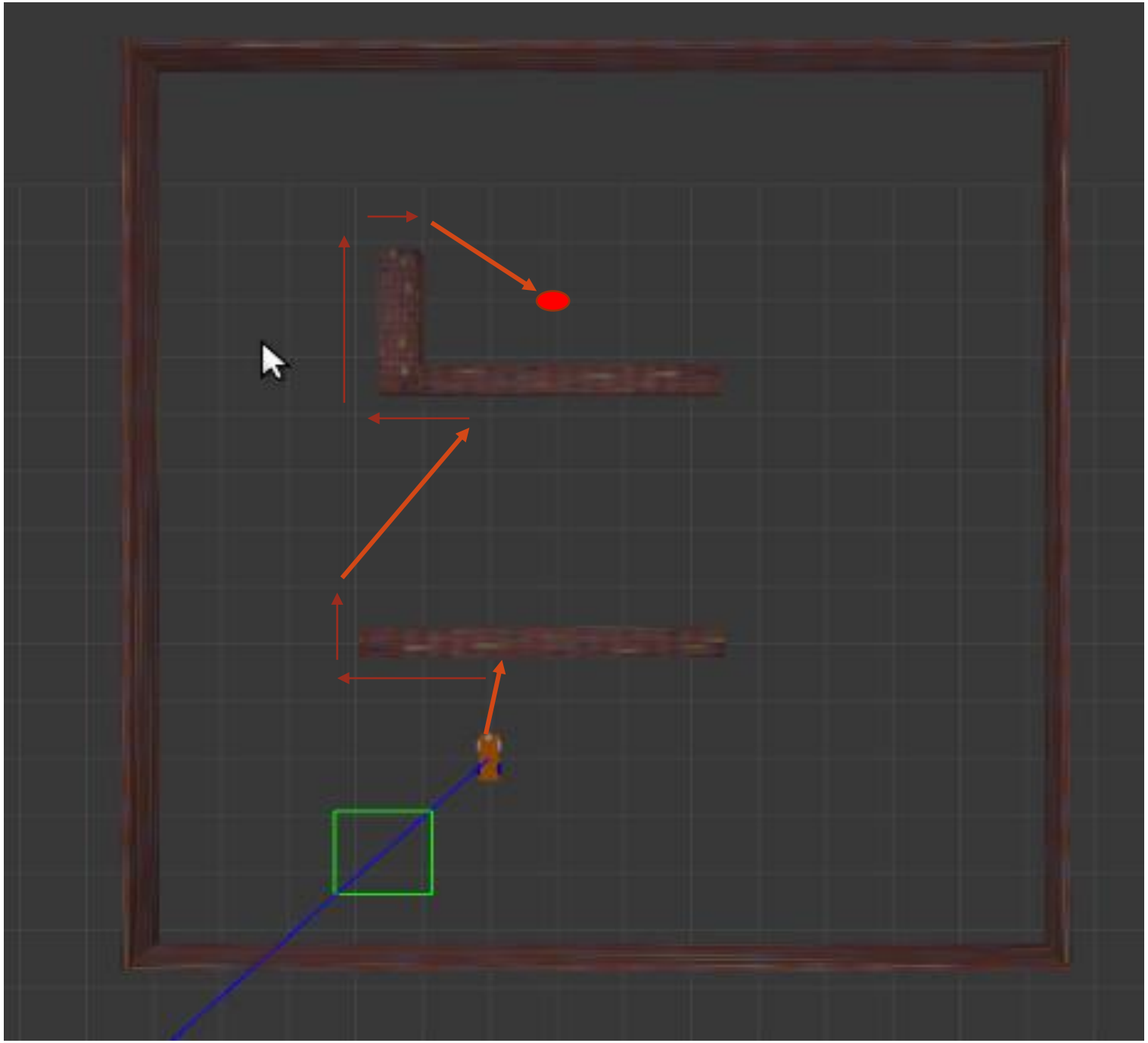
BUG 0 ALGORITHM



Goal point

Wall Follower

Go to point

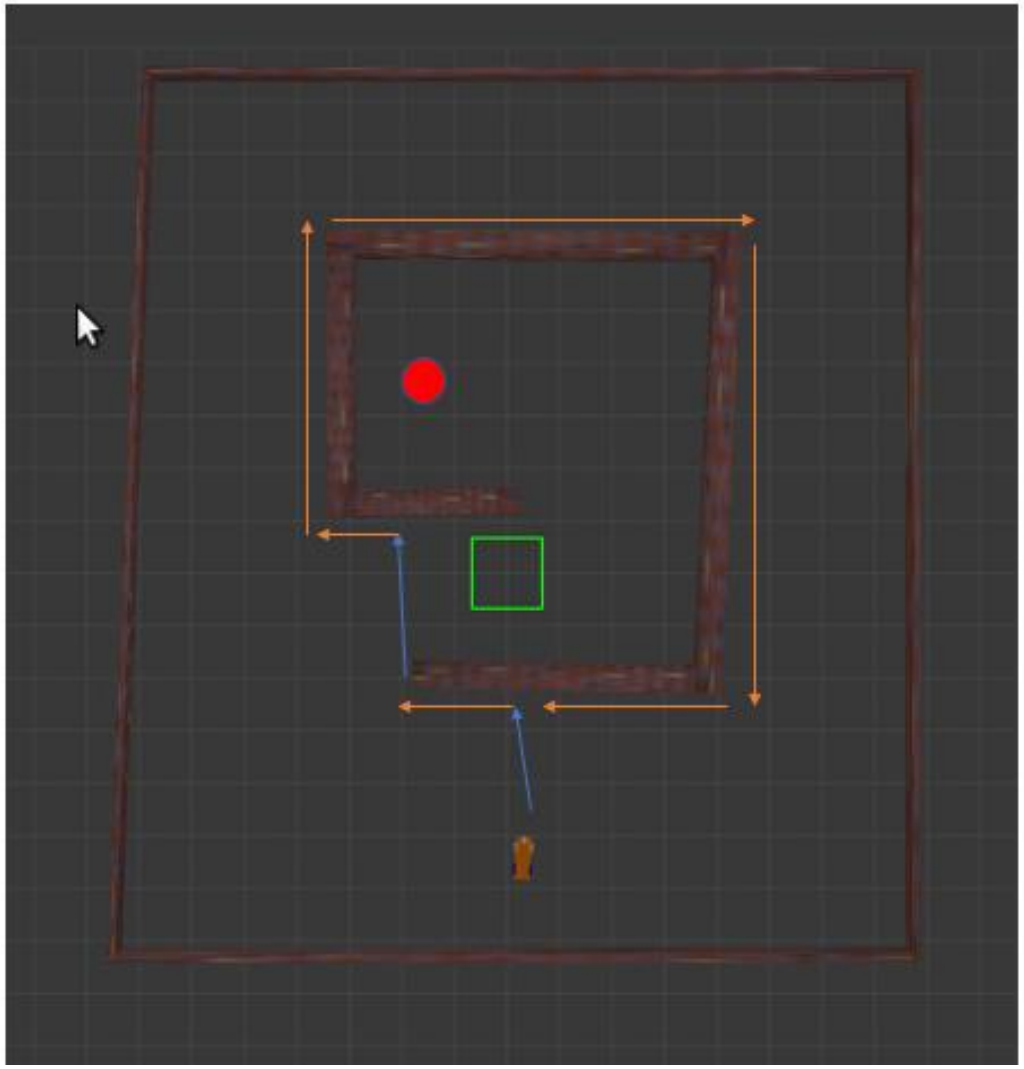


BUG 0 FAILURE

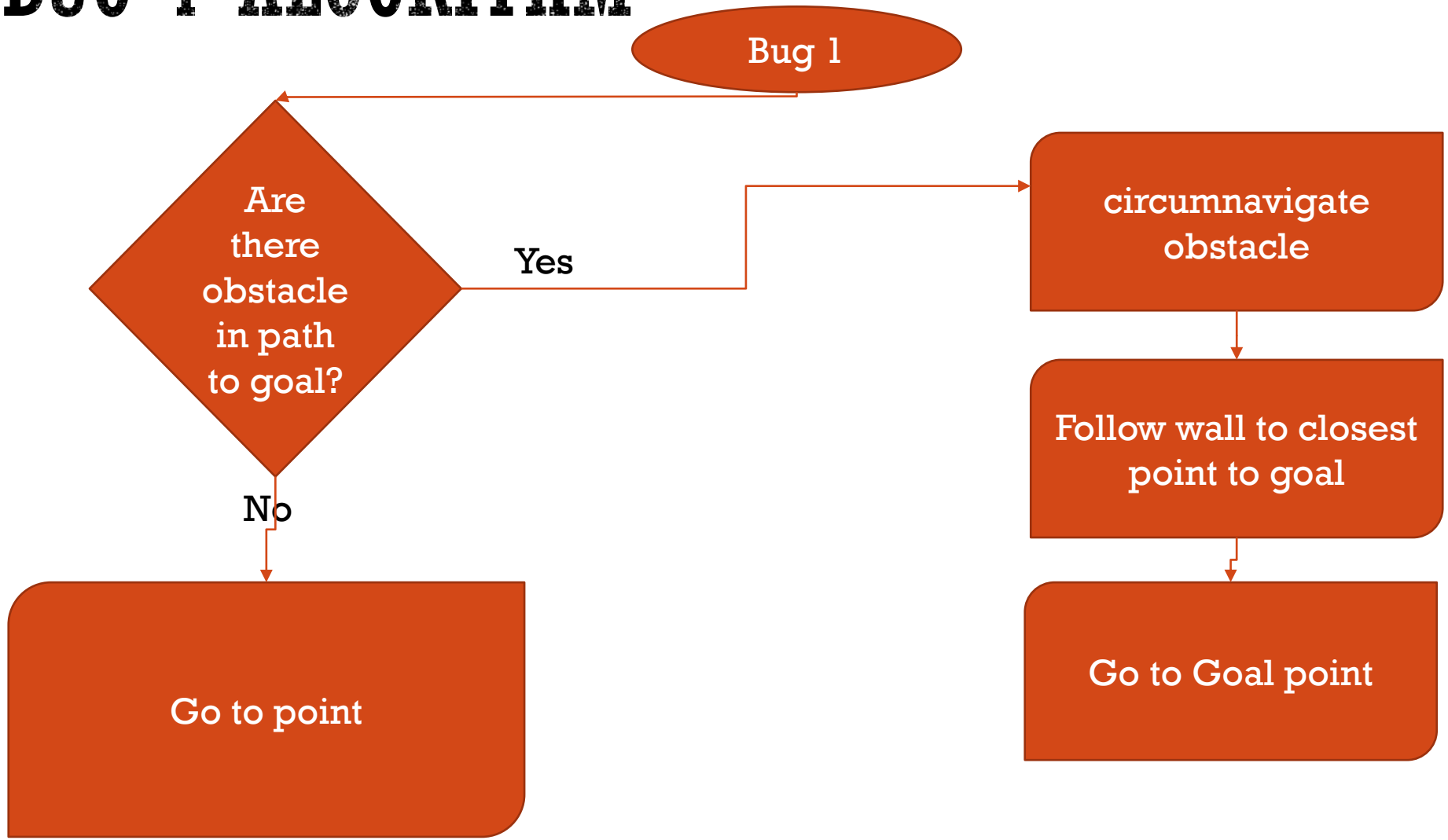
Goal

Go to point

Follow wall



BUG 1 ALGORITHM



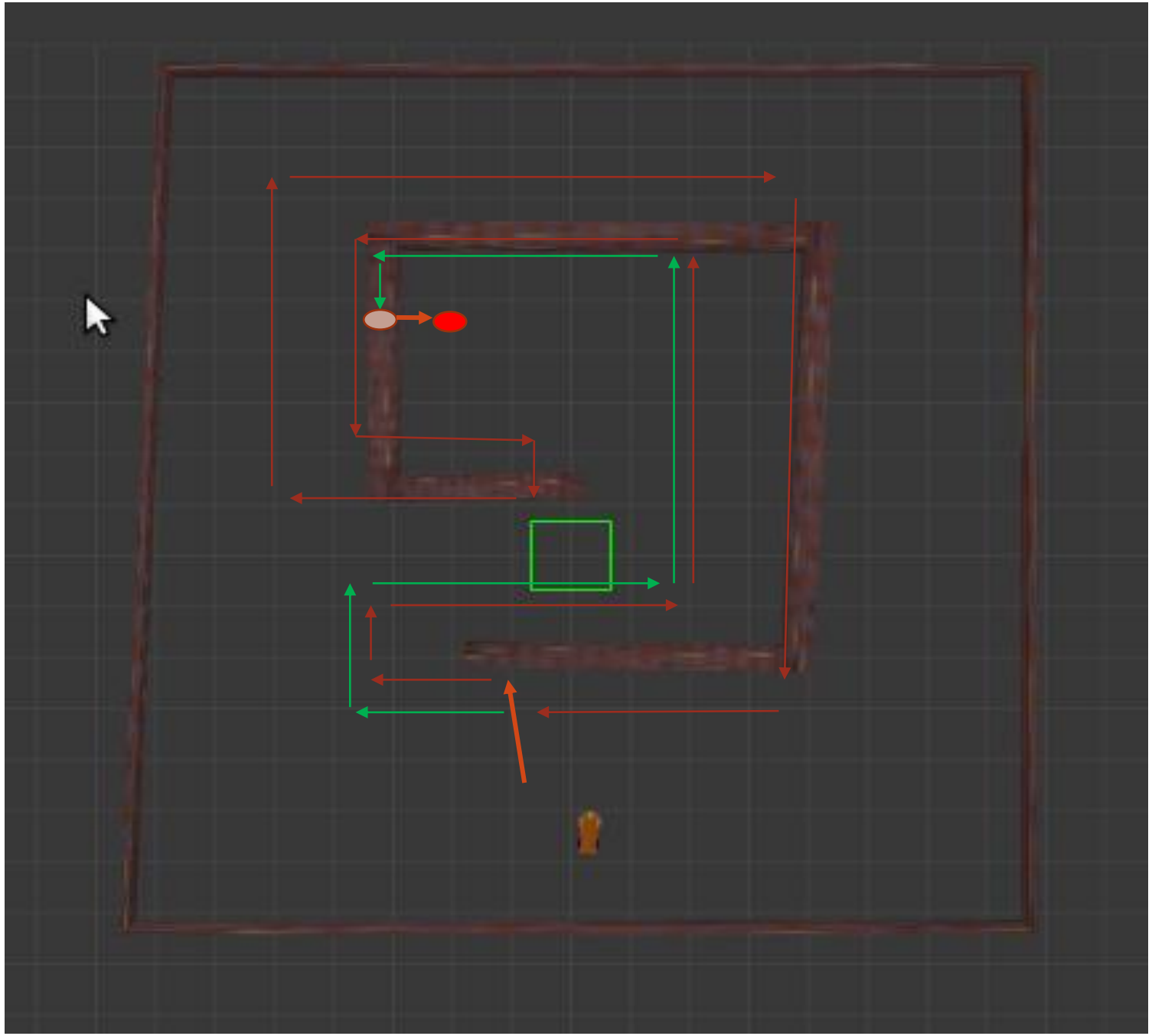
Goal point

Closest point

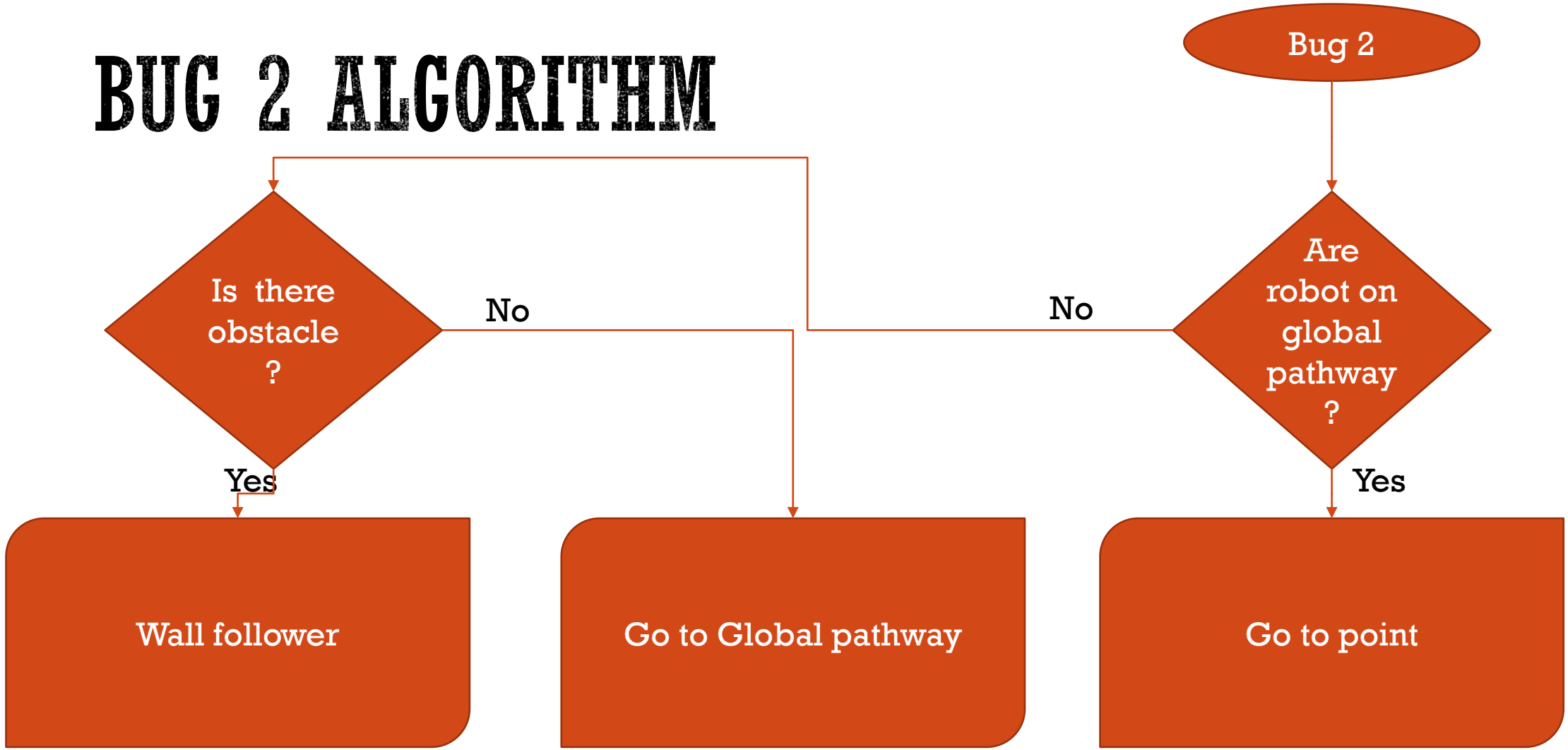
Go to point

circumnavigate

Go to closest point



BUG 2 ALGORITHM



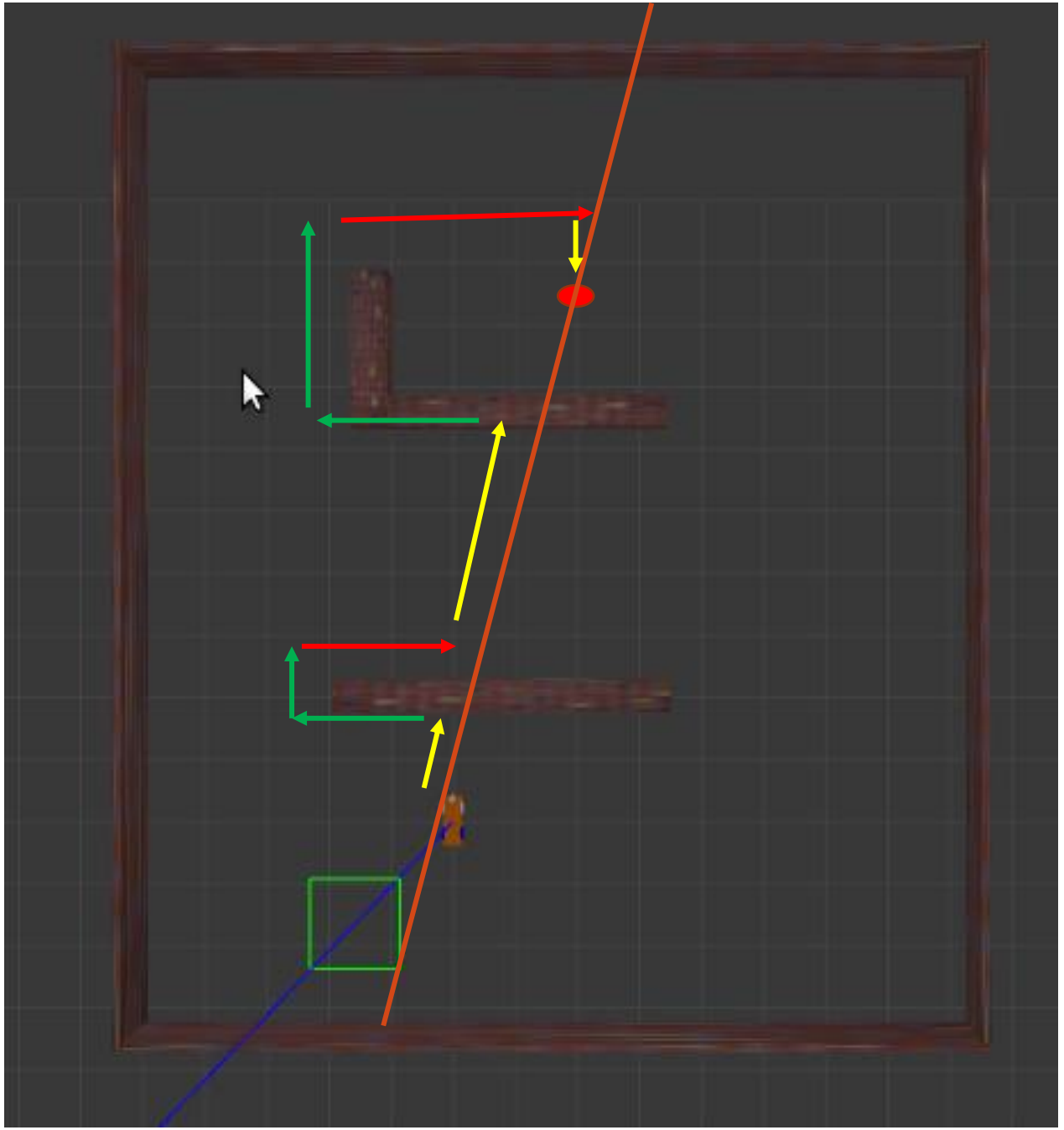
Goal point

Global pathway

Go to point

Follow wall

Go to pathway



CONCLUSION

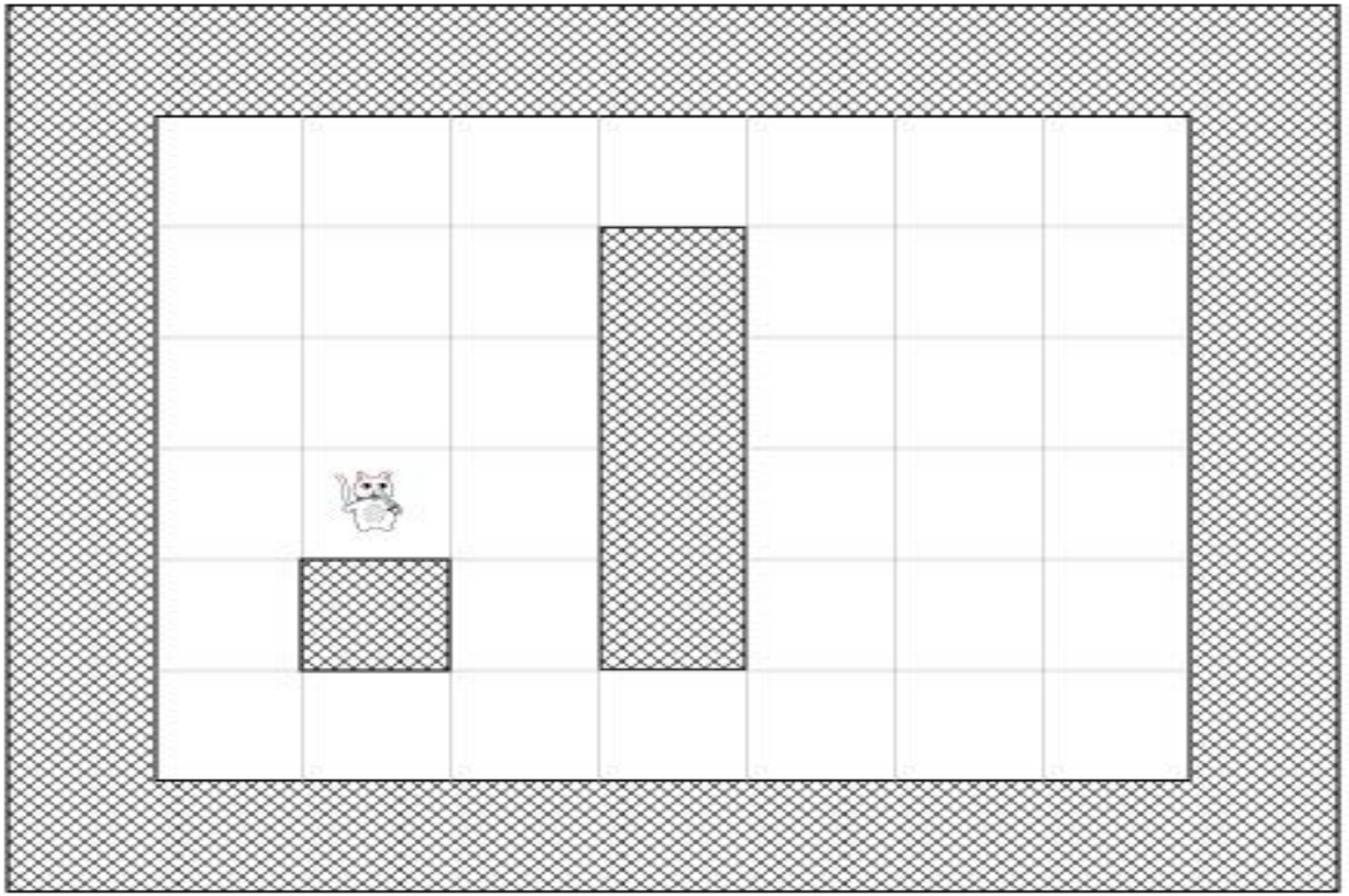
- Bugs algorithms are not efficient to use for autonomous vehicles like rovers, as it shown before from test cases, so we need for more advanced algorithms to depend on it in automation of our rover but also bugs algorithms are not useless

exploration

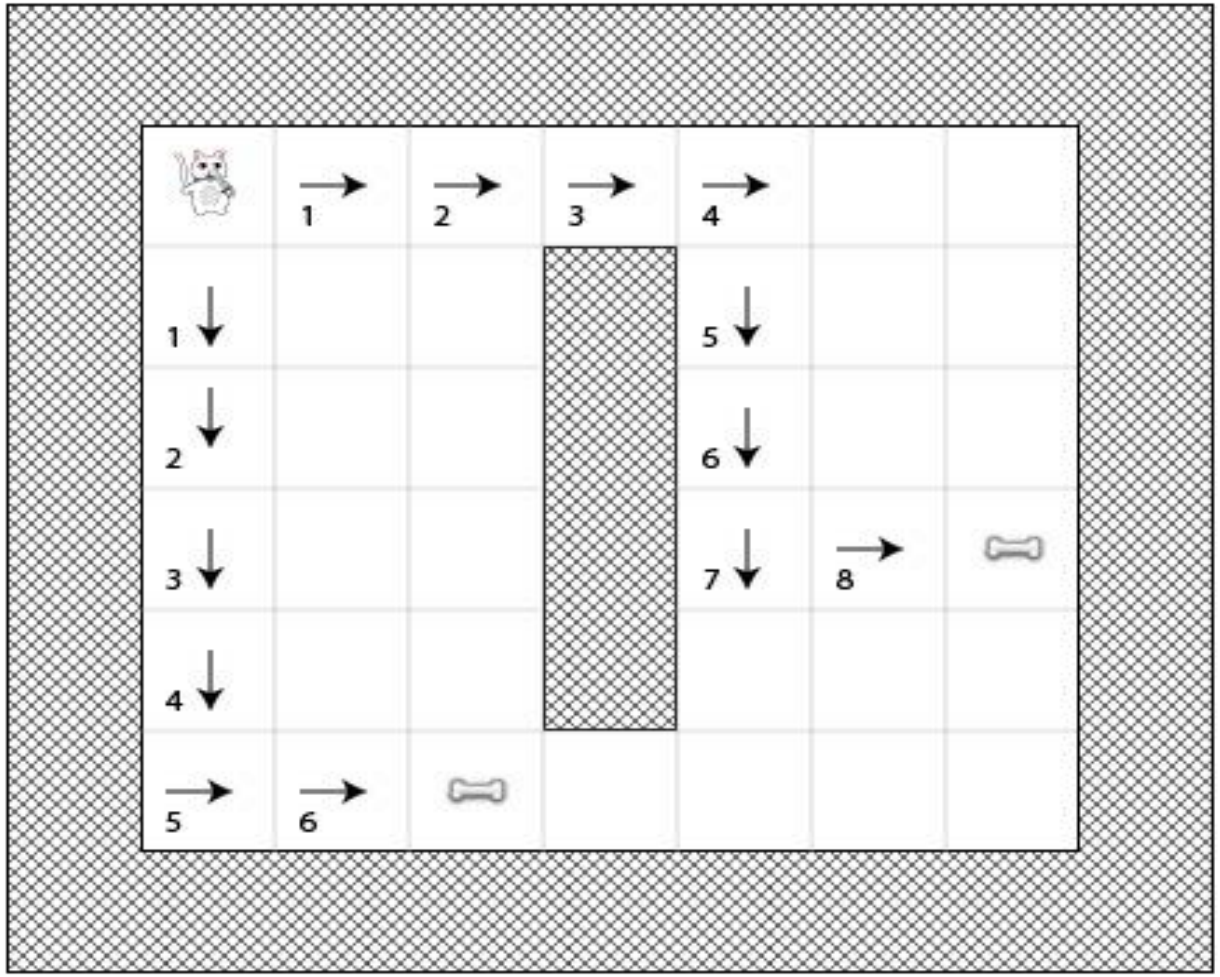
A STAR ALGORITHM

- F(n) = g(n) + h(n)
- F : total cost to reach next node
- g : the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.
- h : the estimated movement cost to move from that given square on the grid to the final destination

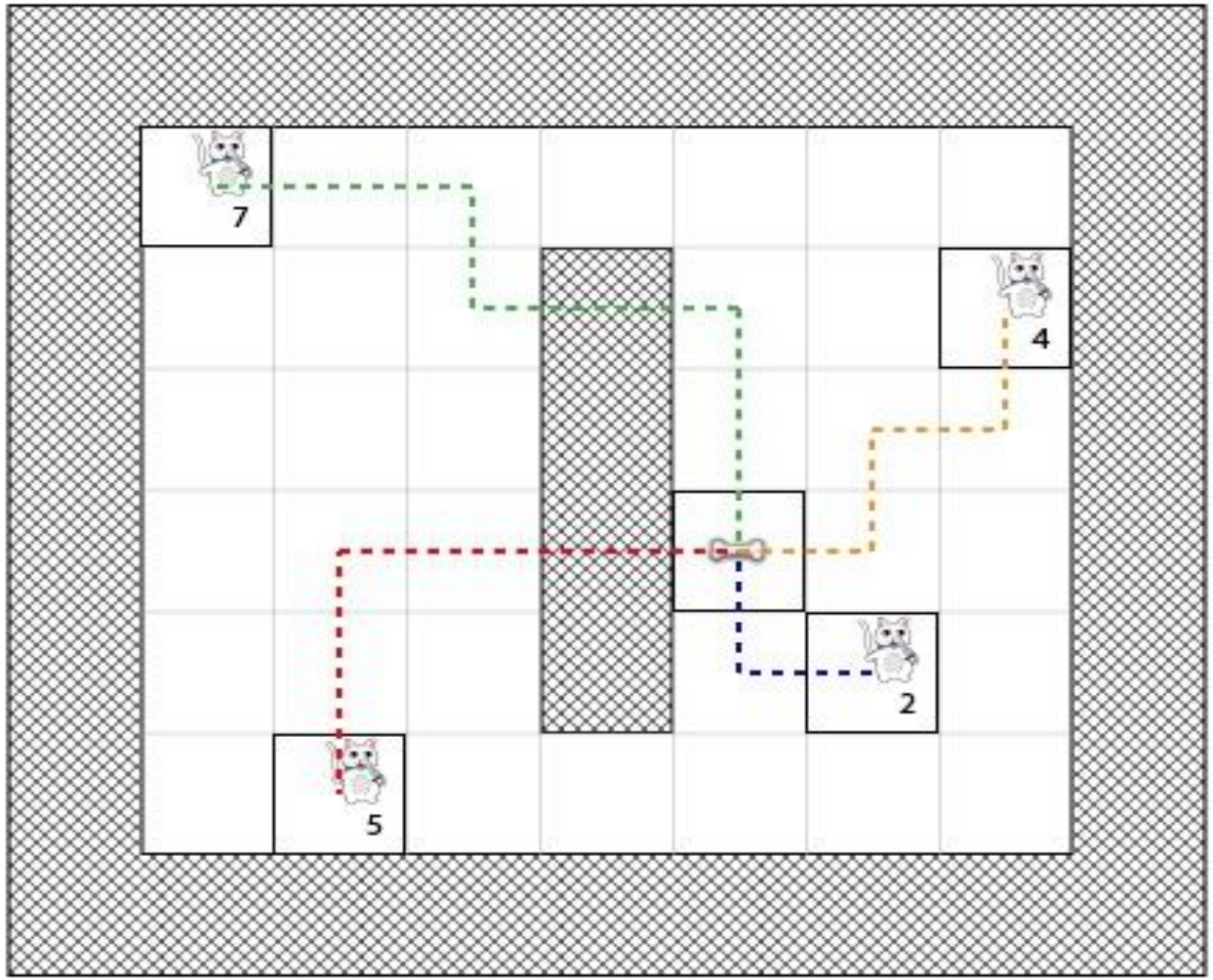
A* TUTORIAL "GRID"



G CALCULATION



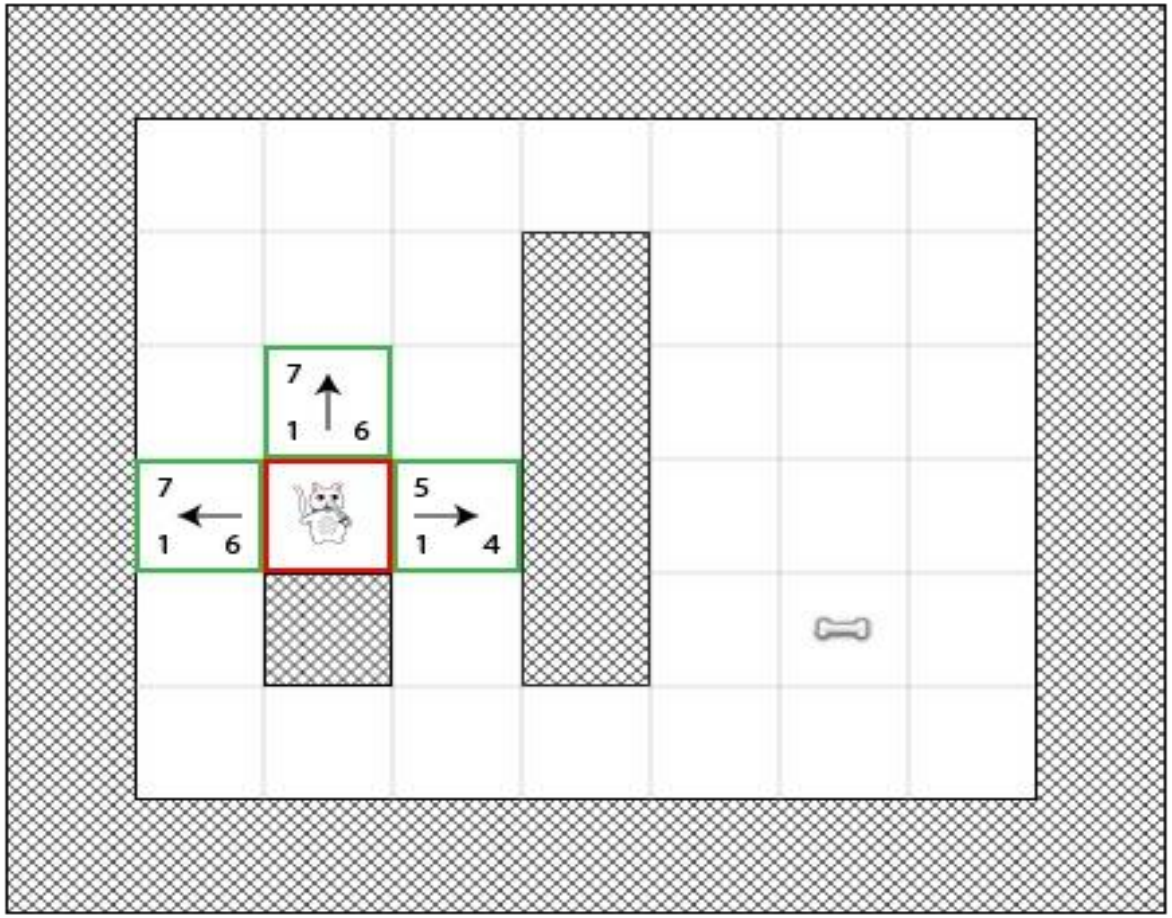
H CALCULATION "MANHATTAN LENGTH"

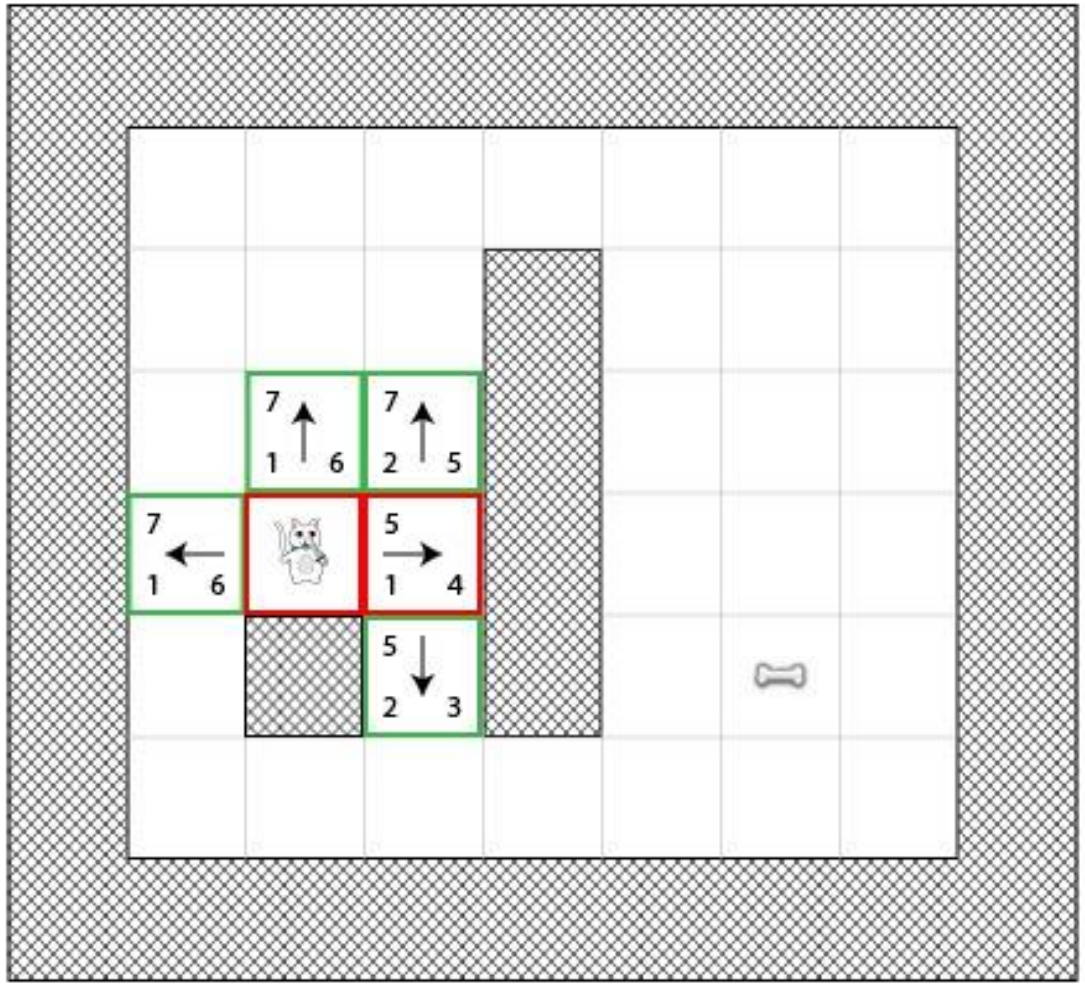


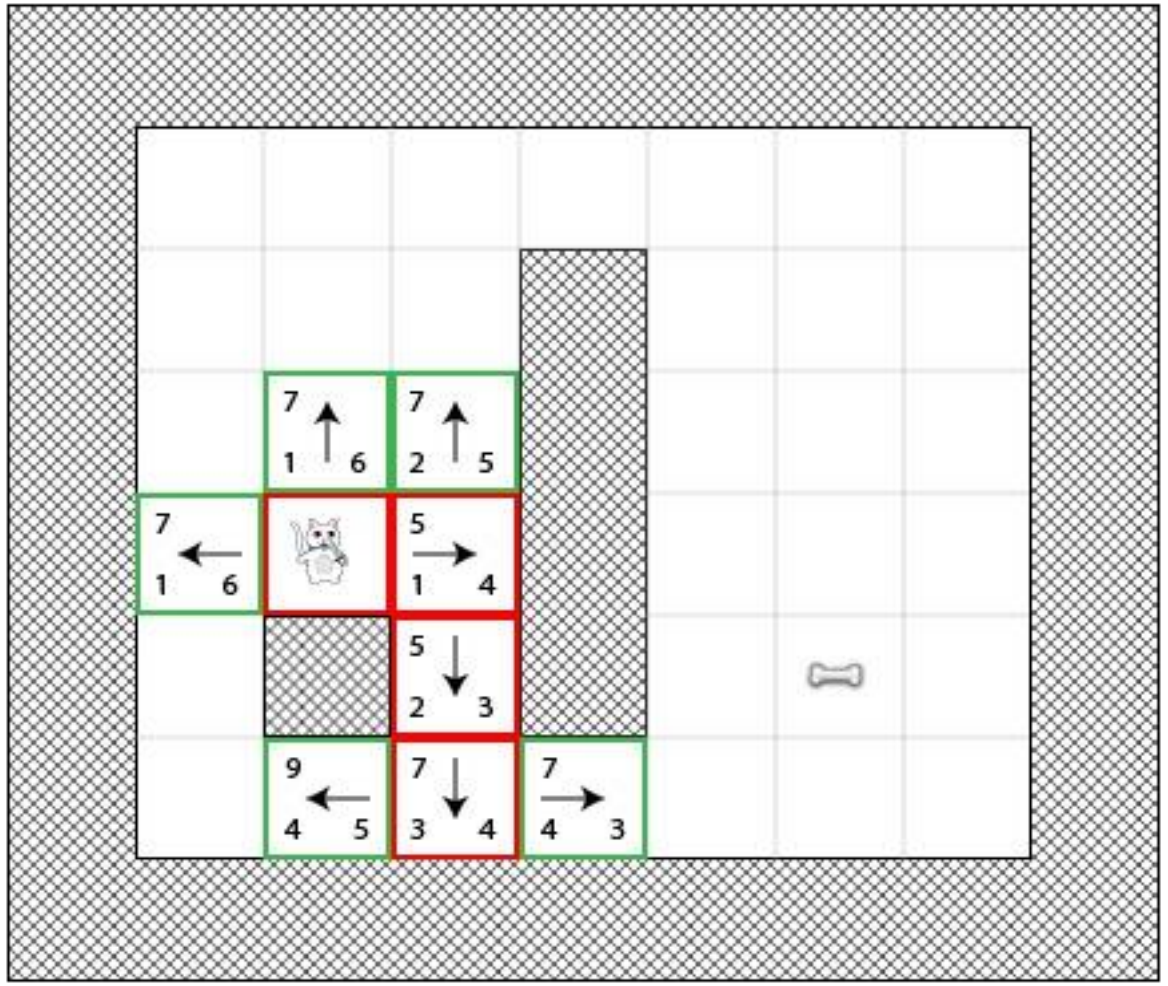
TEST CASE "CAT TO BONE"

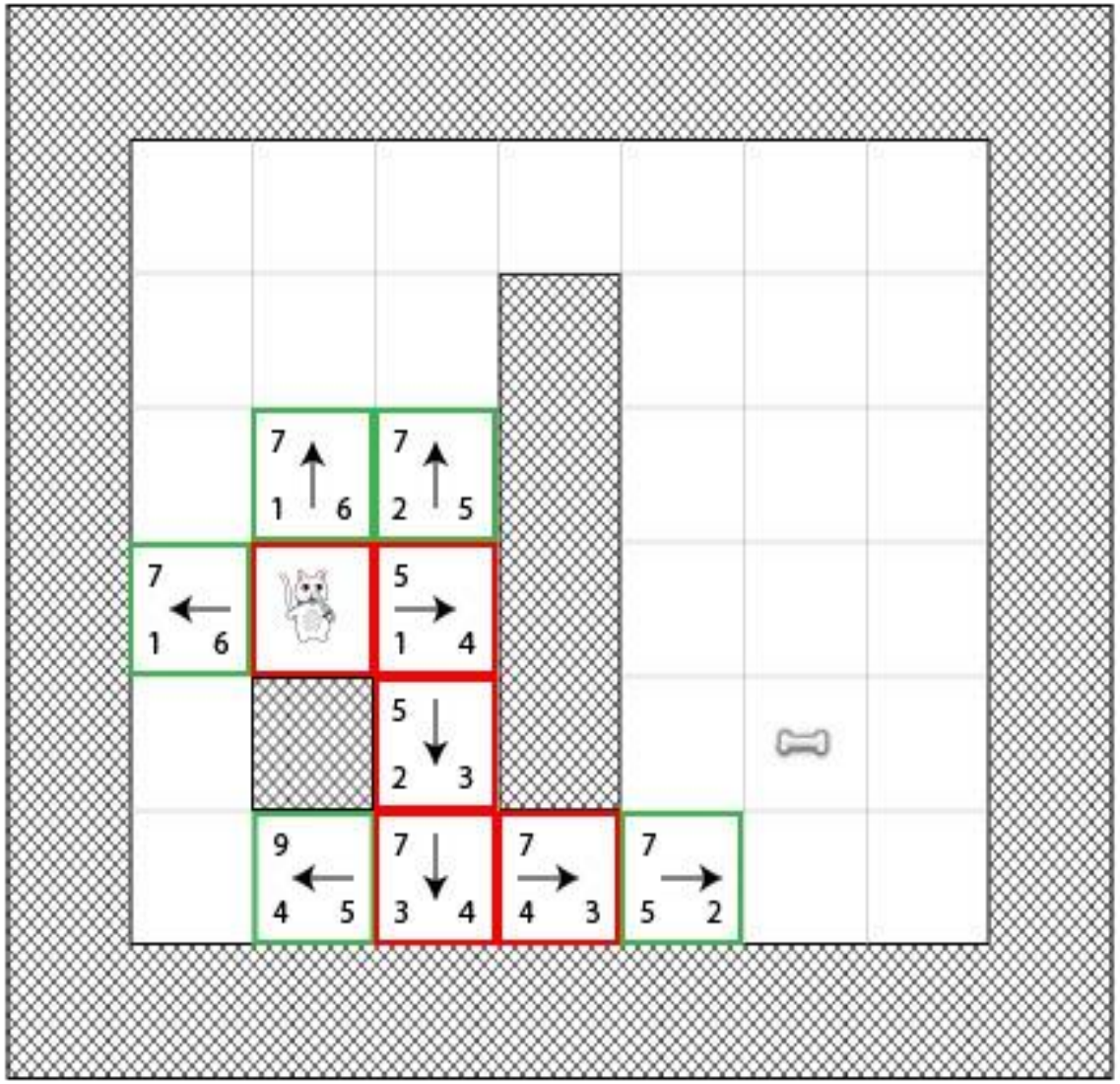
F
G H

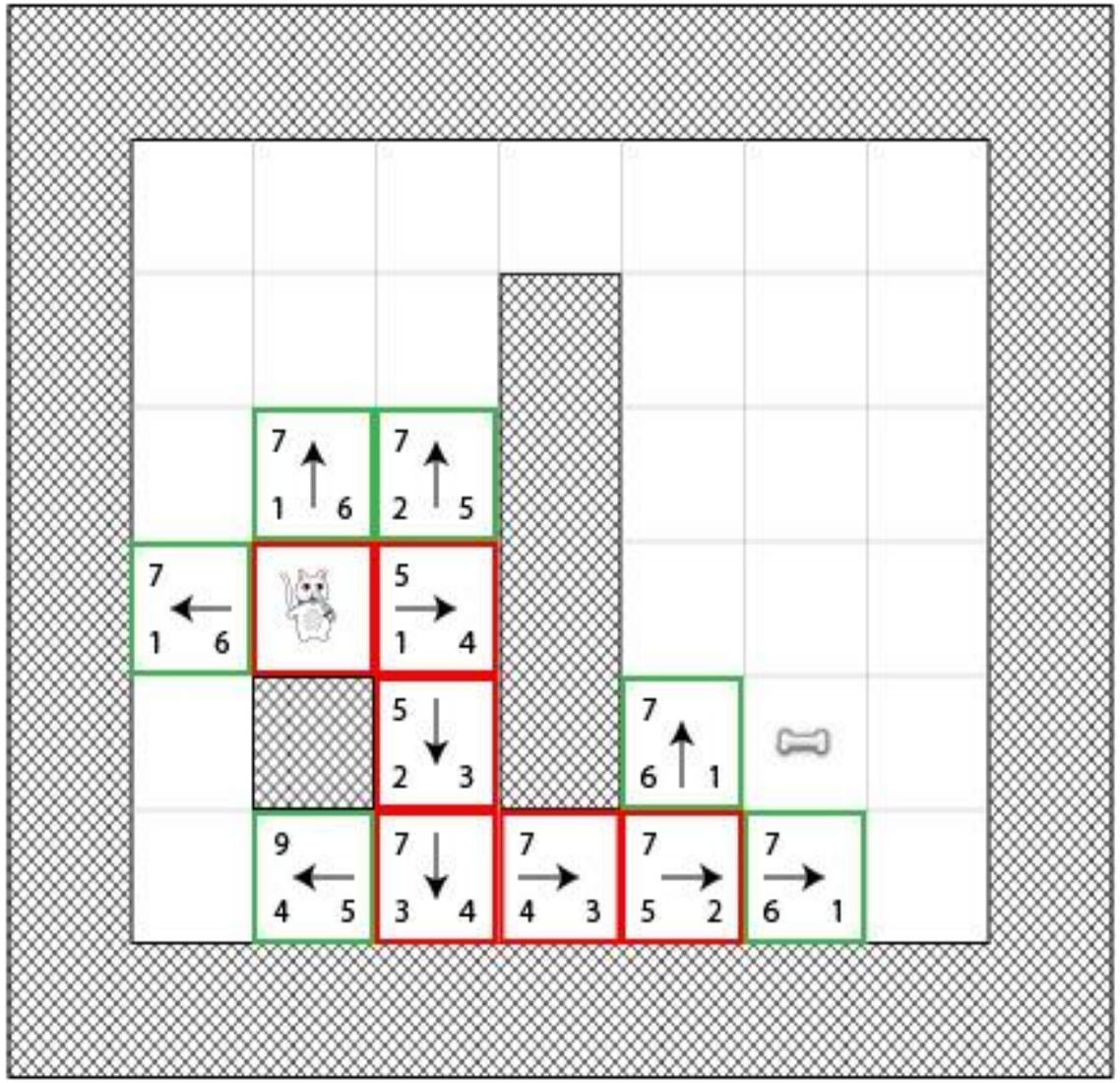
Key




7
1 6
7
2 5
7
1 6
5
1 4
5
2 3

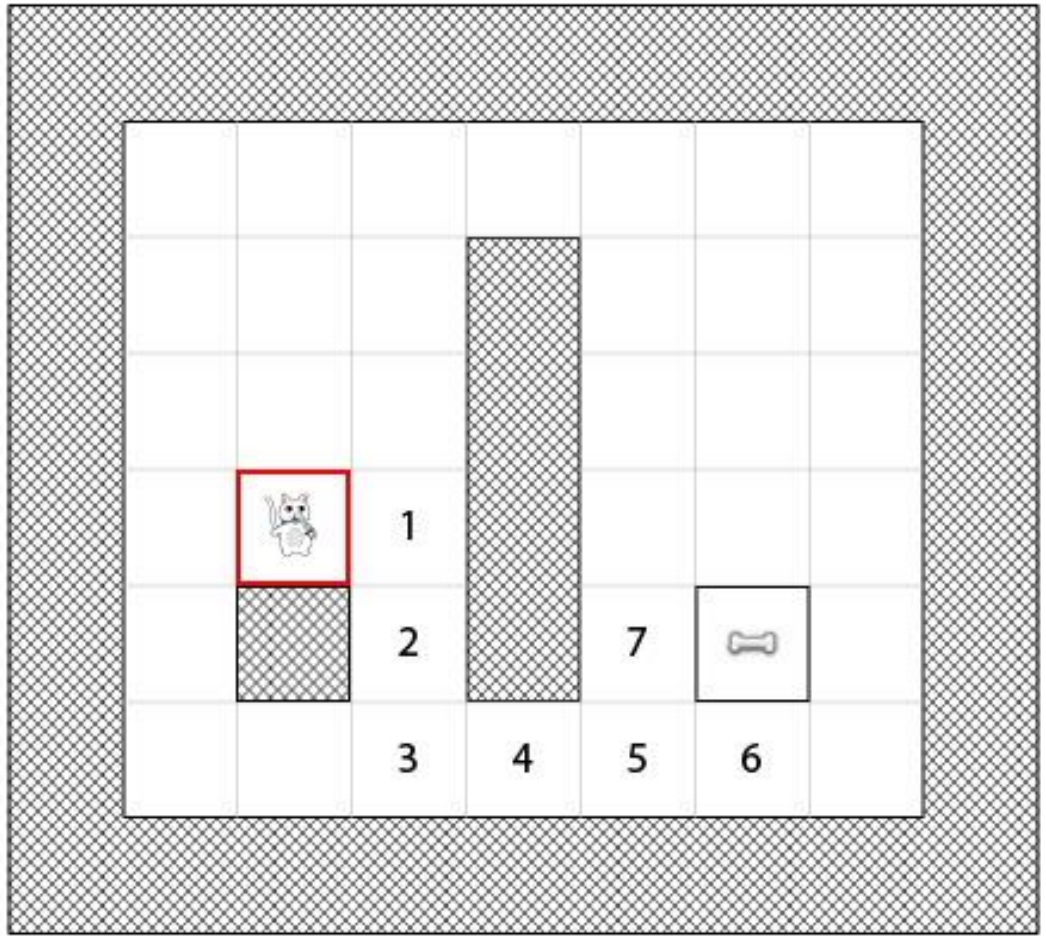

7
1 6
7
2 5
7
1 6
5
1 4
5
2 3
9
4 5
7
3 4
7
4 3

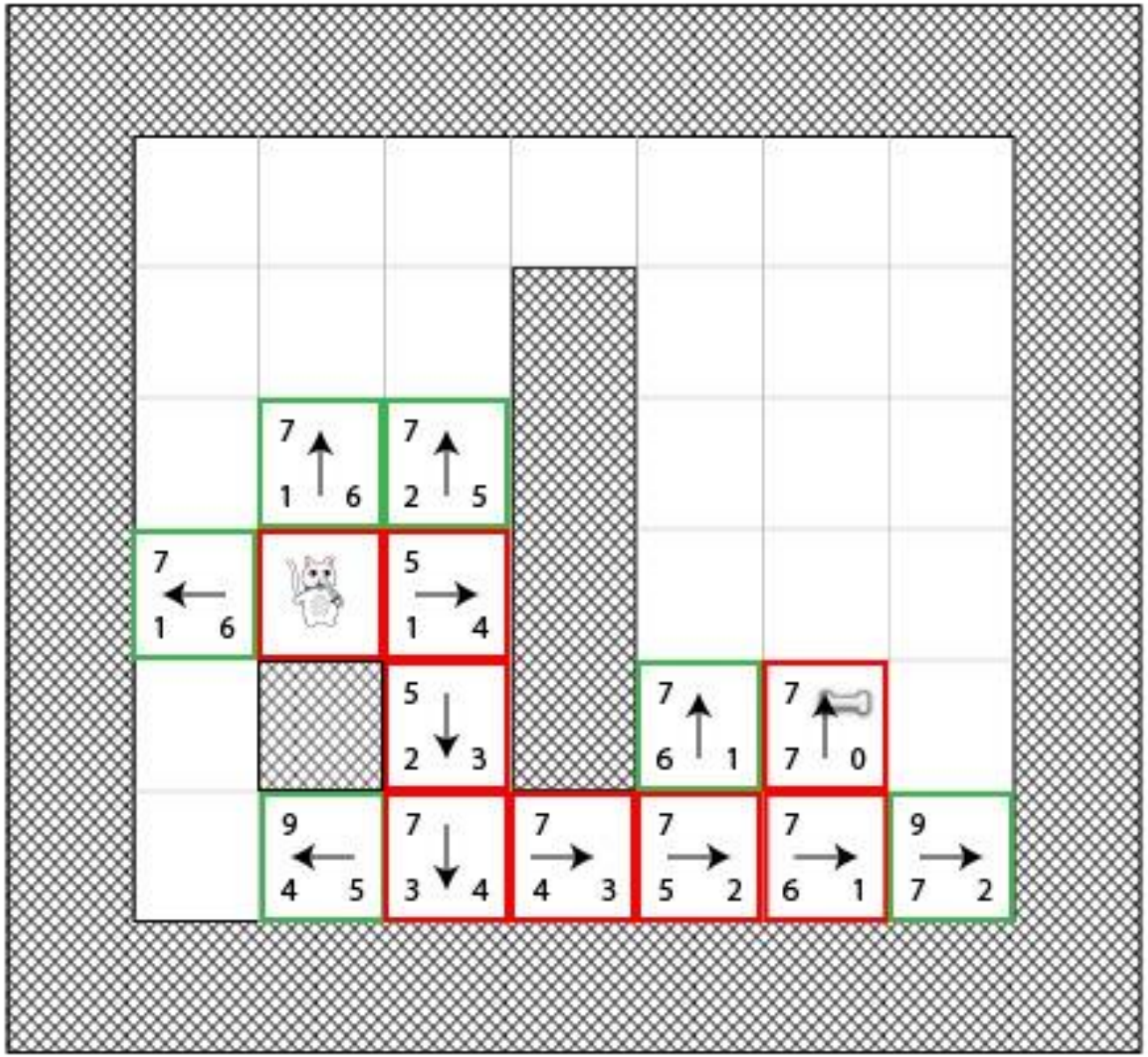

7 1 6
7 2 5
7 1 6
5 1 4
5 2 3
9 4 5
7 3 4
7 4 3
7 5 2


7 1 6
7 2 5
7 1 6
5 1 4
5 2 3
7 6 1
9 4 5
7 3 4
7 4 3
7 5 2
7 6 1

The two paths:

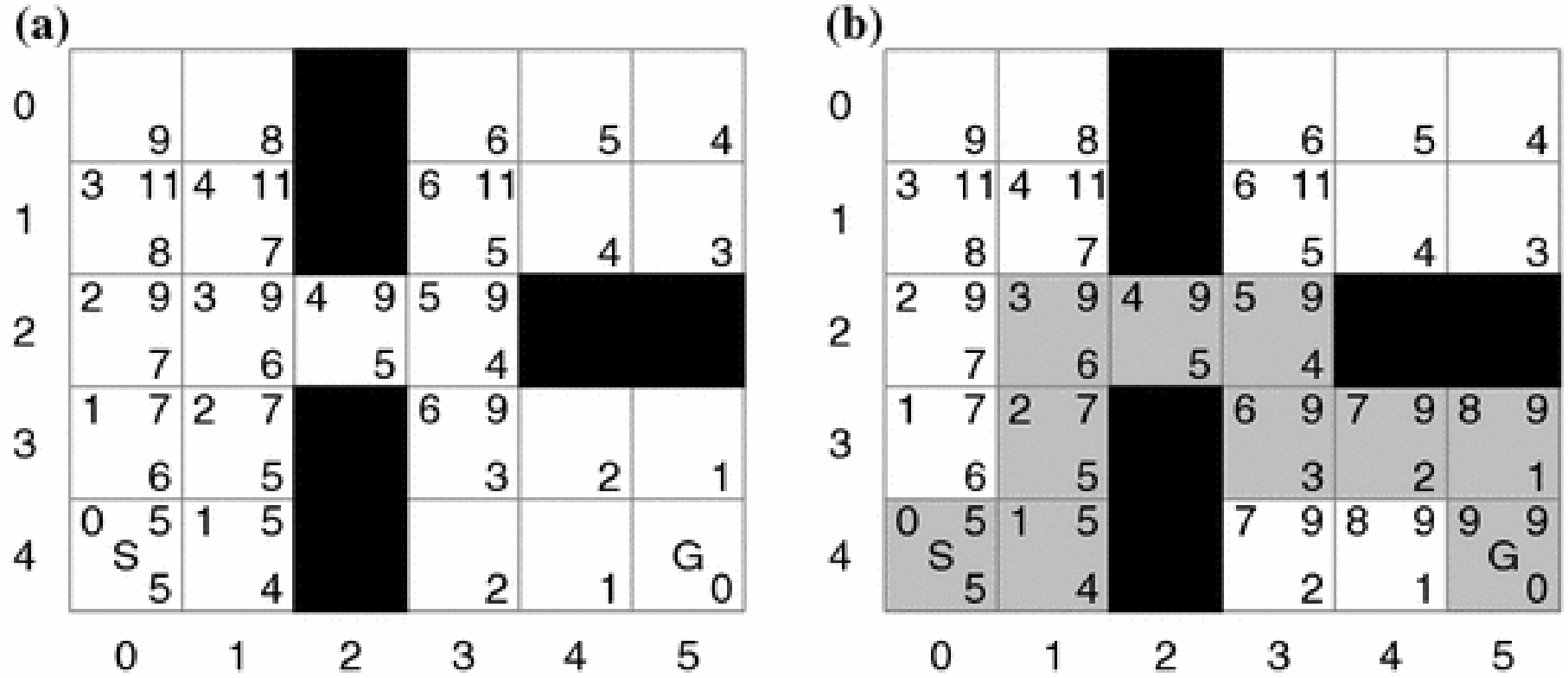
- 1-2-3-4-5-6-Bone
- 1-2-3-4-5-7-Bone




7 1 6
7 2 5
7 1 6
5 1 4
5 2 3
7 6 1
7 7 0
9 4 5
7 3 4
7 4 3
7 5 2
7 6 1
9 7 2

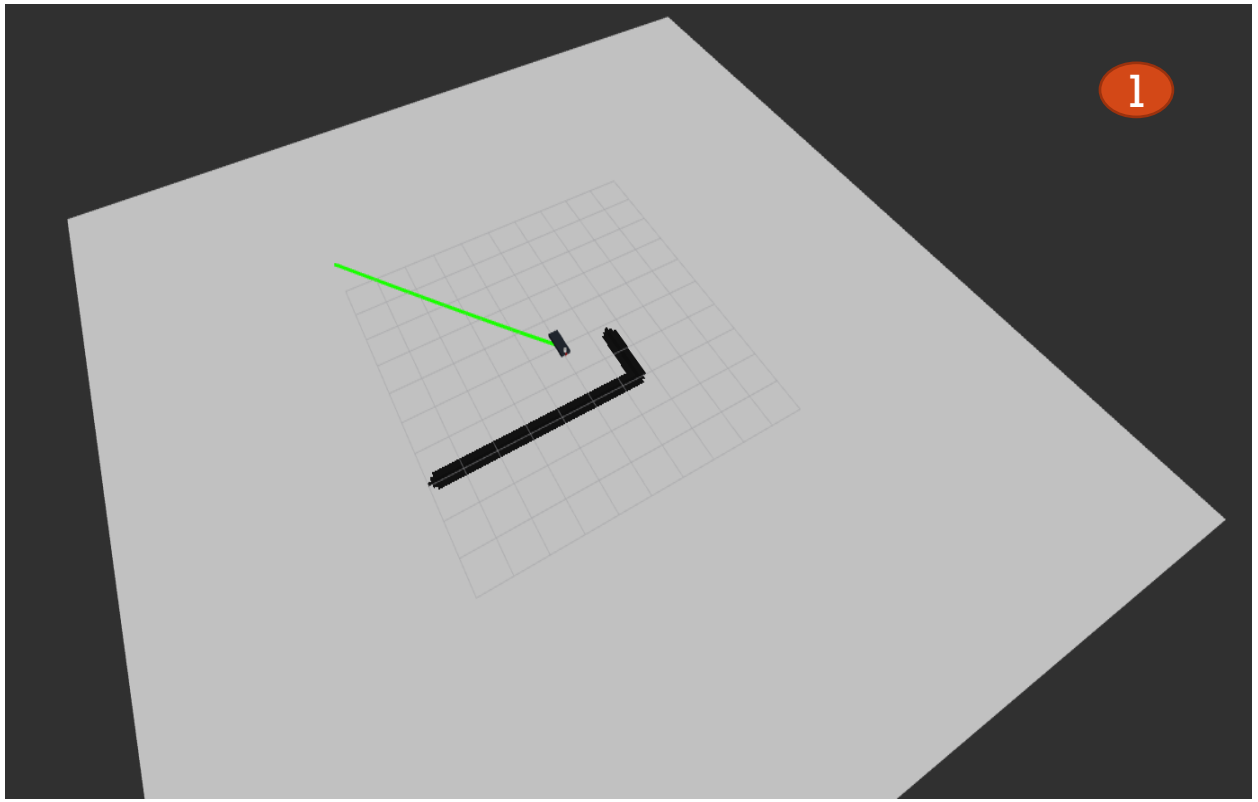
g	f
	h

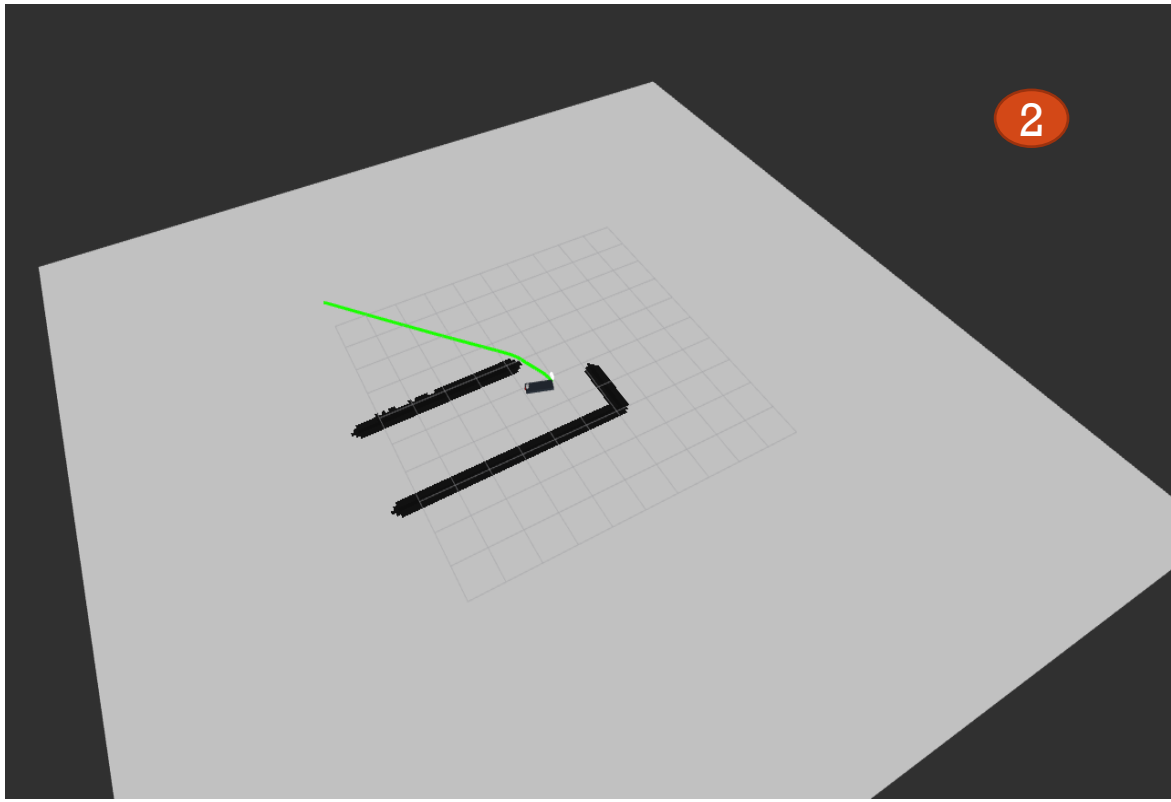
Keys

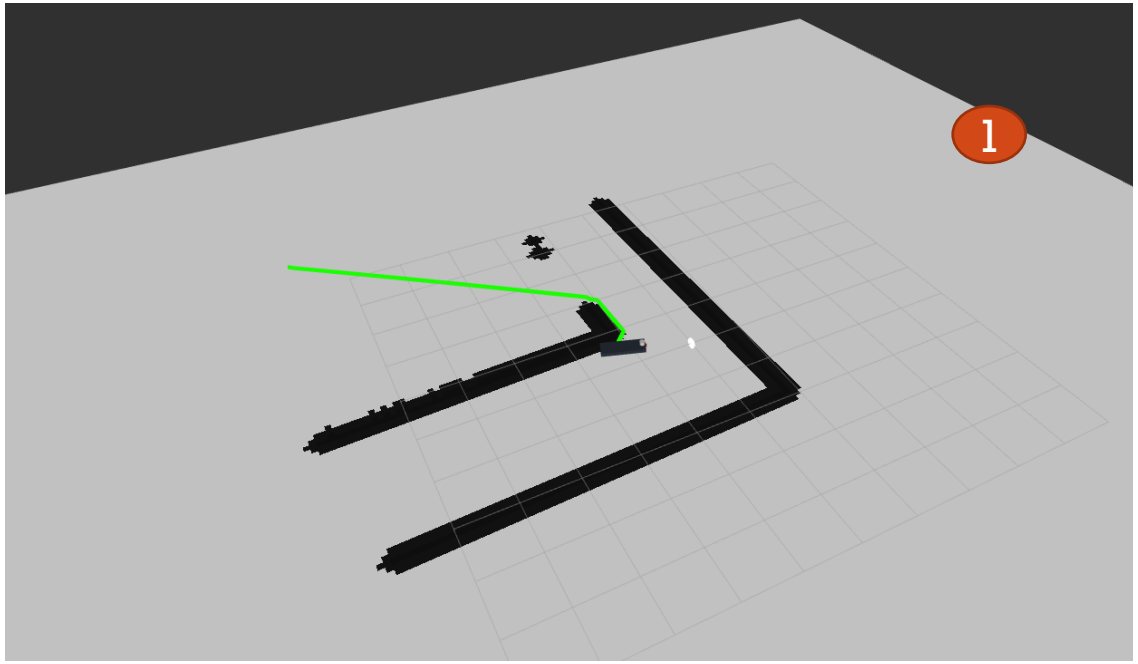


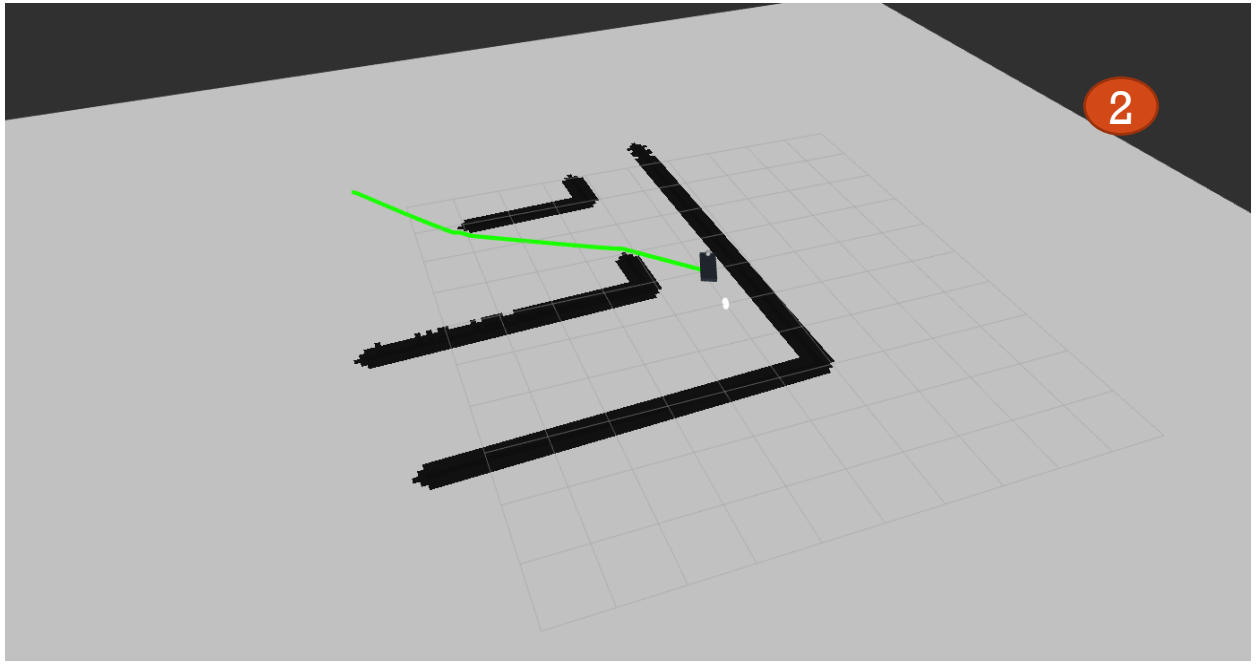
Cost map and pass from start(s) to Goal(G)

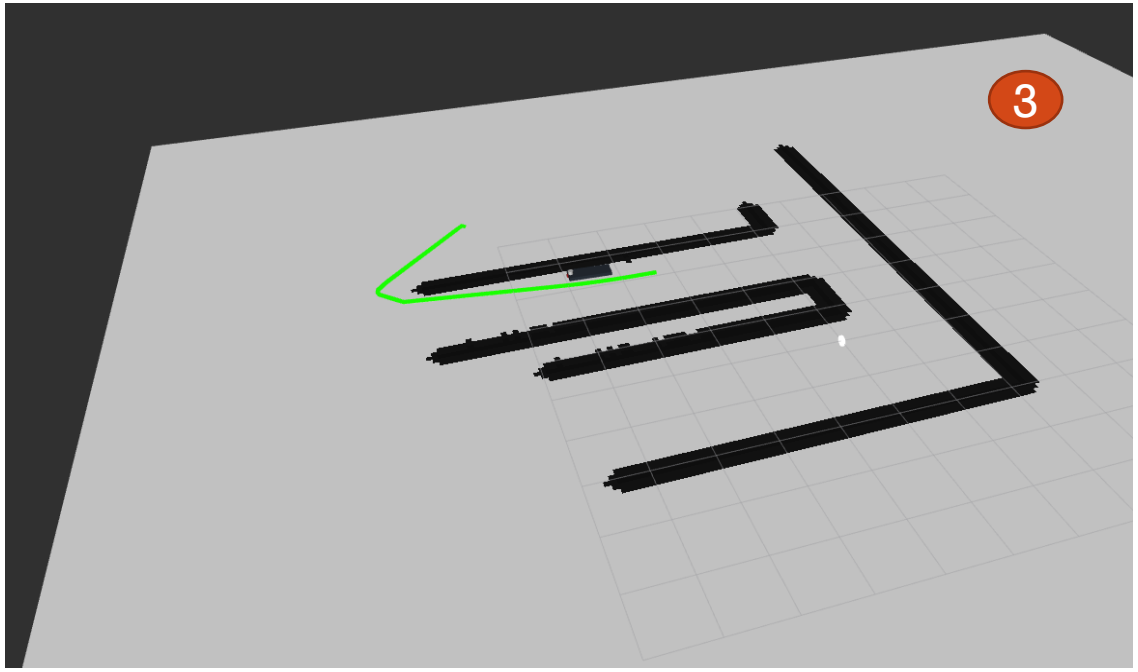
ACTUAL IMPLEMENTATION

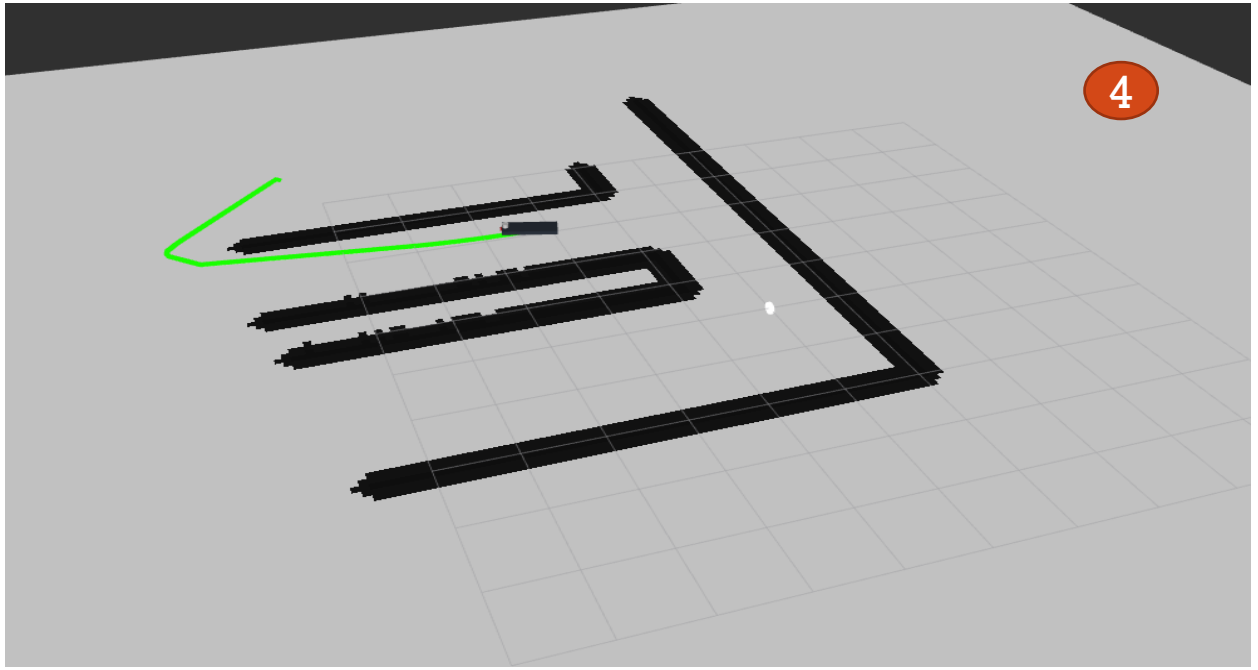


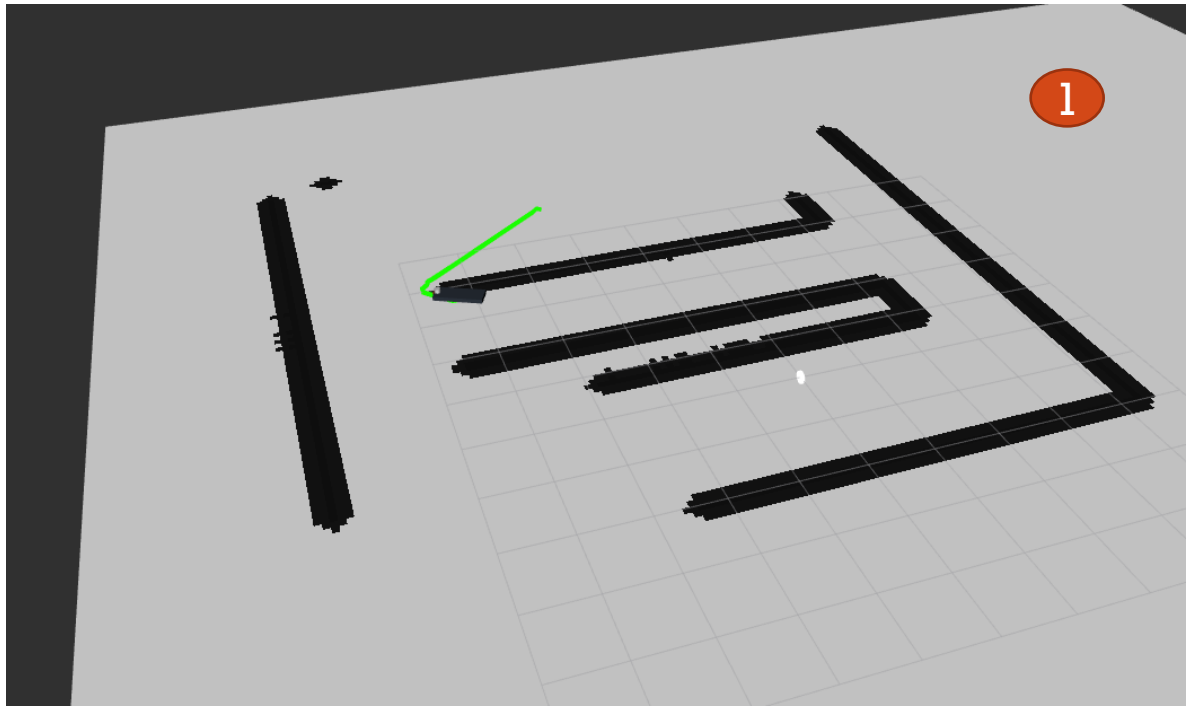


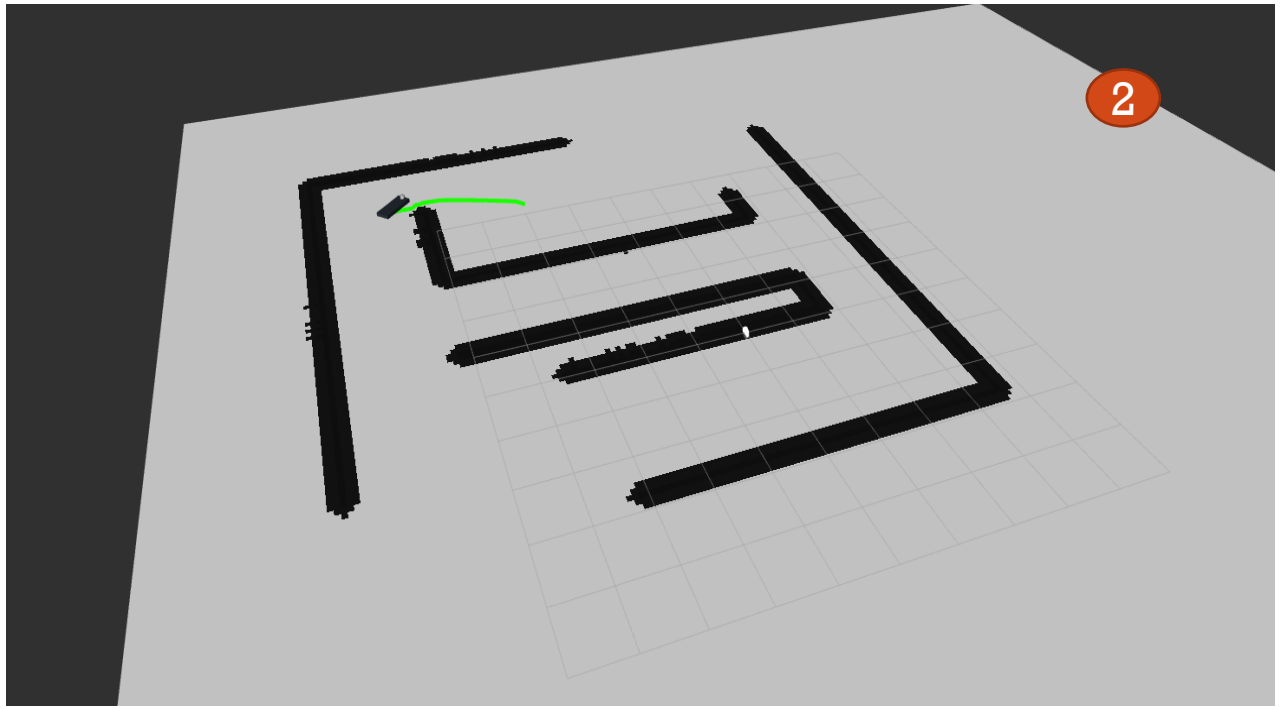

1


2


3


4


1


2