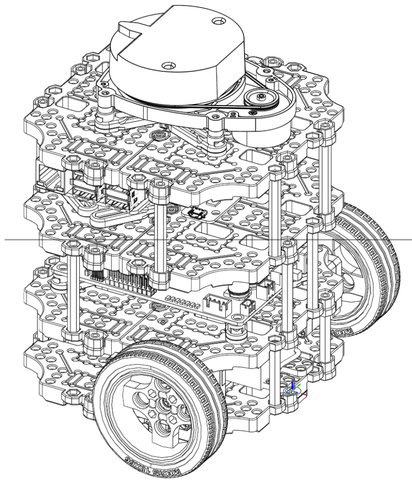


Autonomous Robotic Systems 2023
Lab. Notes

Pedro Martins

https://www.isr.uc.pt/~pedromartins
pedromartins@isr.uc.pt

Institute of Systems and Robotics (ISR)
Department of Electrical and Computer Engineering
University of Coimbra, Portugal

TurtleBot3 MatLab/ROS Interface - Overview

```
% Constructor (init ROS connection)
IP_TURTLEBOT = "192.168.1.xxx";          % virtual machine IP (or robot IP)
IP_HOST_COMPUTER = "192.168.1.xxx";      % local machine IP
tbot = TurtleBot3(IP_TURTLEBOT, IP_HOST_COMPUTER);


% send linear and angular velocity commands to robot
tbot.setVelocity(v, w);

% stop robot
tbot.stop();

% define/overwrite pose of robot
tbot.setPose(x, y, theta);

% reset pose: (x,y,theta) = (0,0,0)
tbot.resetPose();

% read 2D pose (and timestamp)
[x, y, theta, timestamp] = tbot.readPose();

% read LIDAR data
[scanMsg, lddata, ldtimestamp] = tbot.readLidar();

% read (simulated) encoders data
[dsr, dsl, pose2D, timestamp] = tbot.readEncoders();

% Destructor — deleting (tbot) object will shutdown ROS connection
clear tbot;
```

doc TurtleBot3

Query system

help TurtleBot3.TurtleBot3
help TurtleBot3.getVersion
help TurtleBot3.getWheelBaseline
help TurtleBot3.getWheelRadius
help TurtleBot3.getGazeboMapName
help TurtleBot3.getNumberTicksPerRevolution
help TurtleBot3.isRealRobot
help TurtleBot3.getBatteryLevel
help TurtleBot3.delete

Robot control

help TurtleBot3.setVelocity
help TurtleBot3.stop
help TurtleBot3.setPose
help TurtleBot3.resetPose

Read sensors

help TurtleBot3.readPose
help TurtleBot3.getPose
help TurtleBot3.initEncoders
help TurtleBot3.readEncoders
help TurtleBot3.getEncodersData
help TurtleBot3.readEncodersWithNoise
help TurtleBot3.getEncodersDataWithNoise
help TurtleBot3.readEncodersTicks
help TurtleBot3.getEncodersDataTicks
help TurtleBot3.readLidar
help TurtleBot3.getLidarData
help TurtleBot3.getInRangeLidarDataIdx
help TurtleBot3.getOutRangeLidarDataIdx
help TurtleBot3.readIMU
help TurtleBot3.getIMUData
help TurtleBot3.readCompass
help TurtleBot3.getCompassData

Gazebo control

help TurtleBot3.gazeboPlace3DCube
help TurtleBot3.gazeboPlace3DCylinder
help TurtleBot3.gazeboPlace3DCardboardBox
help TurtleBot3.gazeboDeleteAllModels
help TurtleBot3.gazeboPause
help TurtleBot3.gazeboUnPause
help TurtleBot3.gazeboResume

Draw Tools

help drawTurtleBot
help drawPolarHistogram
help drawErrorElipse

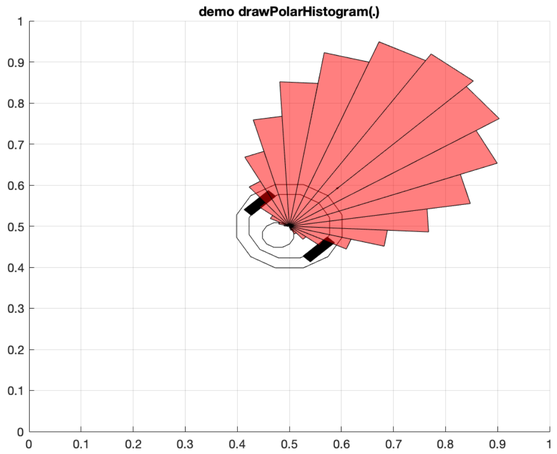
TurtleBot3 software updates

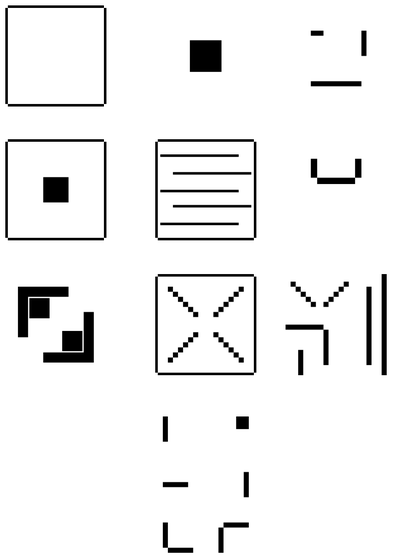
- v08 (lab #1)
- v09 (lab #2)
- v09b
- v09c (lab #3)
- v09d

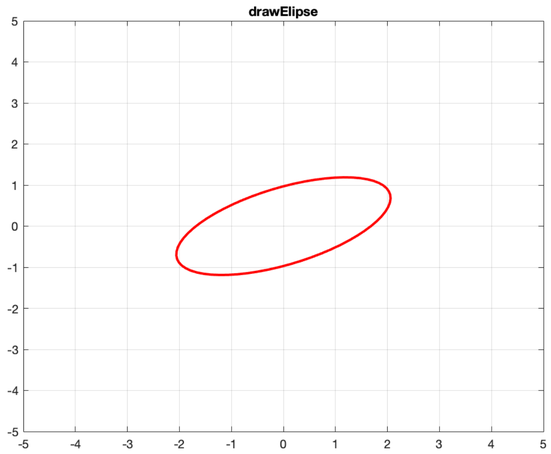
TurtleBot3 software update (#v09)

- Updated constructor format.
- **New 2D/3D maps (#10).**
- **drawPolarHistogram(.).**
- drawElipse(.).
- Bresenham line algorithm (© Aaron Wetzler).

```
IP_TURTLEBOT = "192.168.1.xxx";          % virtual machine IP (or robot IP)
IP_HOST_COMPUTER = "192.168.1.xxx";      % local machine IP

tbot = TurtleBot3(IP_TURTLEBOT, IP_HOST_COMPUTER);   % init TurtleBot obj
```








```
[hp] = drawPolarHistogram(x, y, theta, h, bins, radius, color);
```

```
[hp] = drawErrorElipse( Cov, mu, level, color, linewidth);
```

TurtleBot3 software update (#v09b)

- Added **getGazeboMapName(.)** function that returns the 3D map basename (map loaded at simulator startup).
  - ➡ Requires to install 3D maps again.
- Improved documentation (support for help/doc directives).
- Visual update to drawTurtleBot(.). Added linewidth option.
- bug fixes (csqmap map config files) and minor details.

doc TurtleBot3

Query system

help TurtleBot3.TurtleBot3
help TurtleBot3.getVersion
help TurtleBot3.getWheelBaseline
help TurtleBot3.getWheelRadius
help TurtleBot3.getGazeboMapName
help TurtleBot3.getNumberTicksPerRevolution
help TurtleBot3.isRealRobot
help TurtleBot3.getBatteryLevel
help TurtleBot3.delete

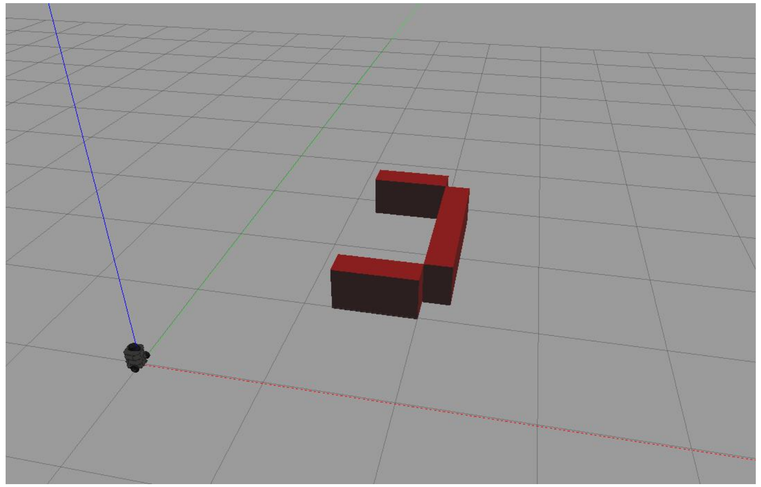
Robot control

help TurtleBot3.setVelocity
help TurtleBot3.stop
help TurtleBot3.readPose
help TurtleBot3.getPose
help TurtleBot3.setPose
help TurtleBot3.resetPose

Read sensors

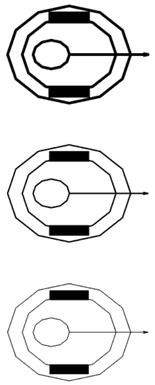
help TurtleBot3.initEncoders
help TurtleBot3.readEncodersData
help TurtleBot3.readEncodersDataWithNoise
help TurtleBot3.readEncodersTicks
help TurtleBot3.readLidar
help TurtleBot3.getLidarData
help TurtleBot3.getInRangeLidarDataIdx
help TurtleBot3.getOutRangeLidarDataIdx
help TurtleBot3.readIMU
help TurtleBot3.getIMUData
help TurtleBot3.readCompass
help TurtleBot3.getCompassData

`getGazeboMapName()`



`'umap'`

`drawTurtleBot()`



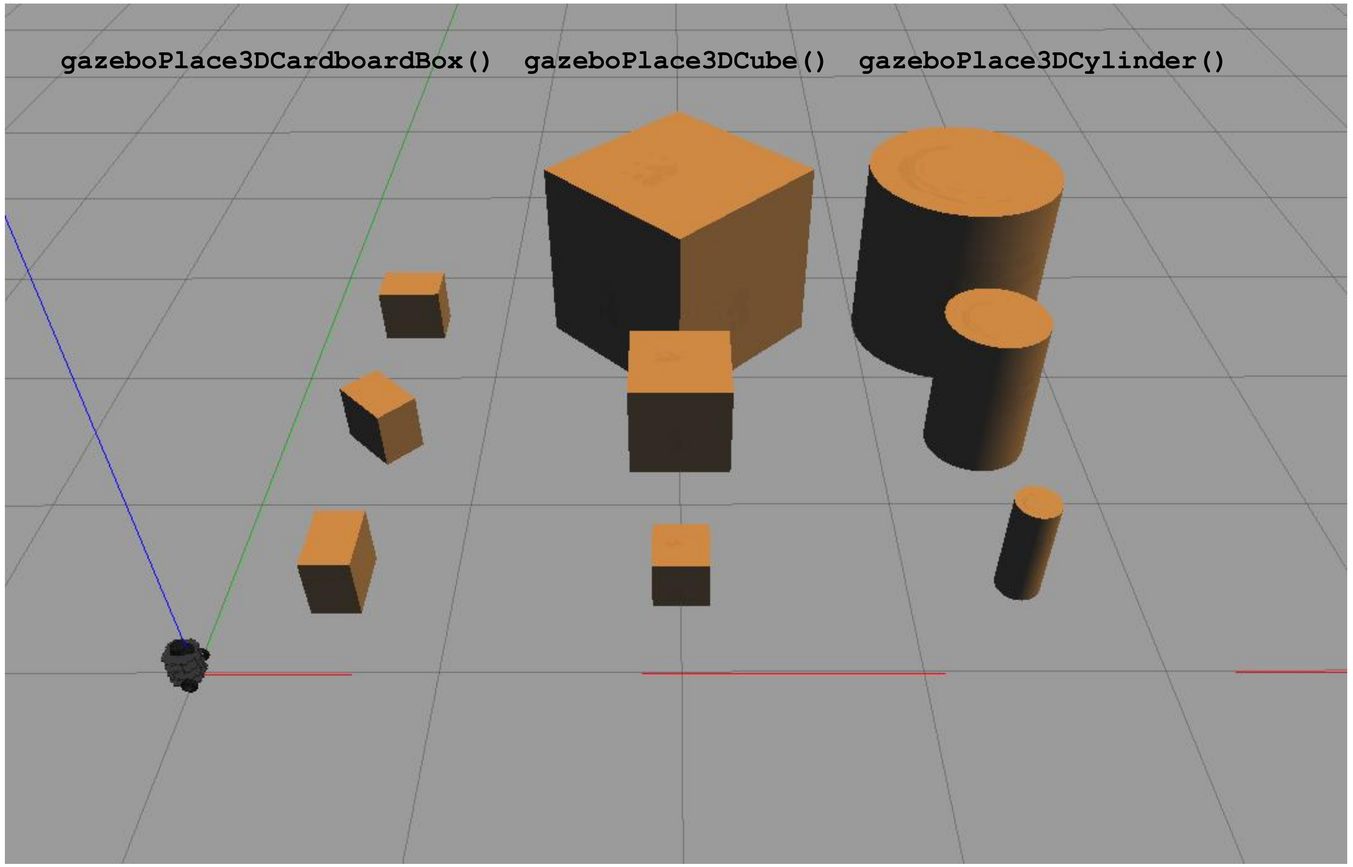
```
image = imread( sprintf('maps/%s_grid1.png', tbot.getGazeboMapName()) );
```

TurtleBot3 software update (#v09c)

- Added Gazebo control functions:
  - gazeboPlace3DCube(x, y, orientation, edgeSize);
  - gazeboPlace3DCylinder(x, y, radius, length);
  - gazeboPlace3DCardboardBox(x, y, orientation);
  - gazeboDeleteAllModels();
  - gazeboPause();
  - gazeboUnPause();

- Cardboard Box model
  - Size (length x width x height): 33 x 22 x 30 cm

help TurtleBot3.gazeboPlace3DCube
help TurtleBot3.gazeboPlace3DCylinder
help TurtleBot3.gazeboPlace3DCardboardBox
help TurtleBot3.gazeboDeleteAllModels
help TurtleBot3.gazeboPause
help TurtleBot3.gazeboUnPause



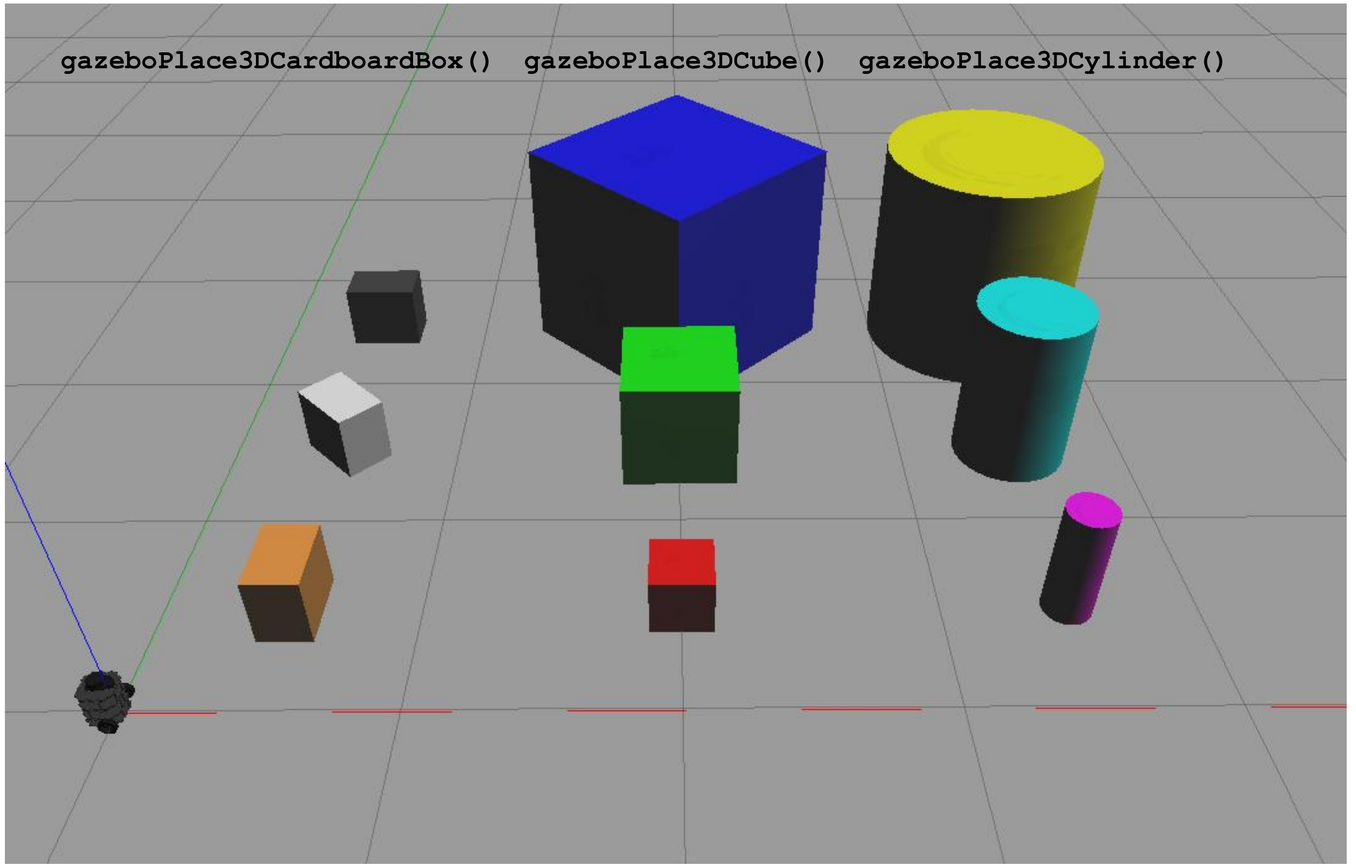
See demoGazeboObjects.m

TurtleBot3 software update (#v09d)

- Updated Gazebo control functions:
  - gazeboResume();
  - Color support in placing objects:
    - gazeboPlace3DCube(.)
    - gazeboPlace3DCylinder(.)
    - gazeboPlace3DCardboardBox(.)

- Updates to LIDAR data index retrieval
  - Added maxRange (optional input argument)
  - [vidx] = tbot.getInRangeLidarDataIdx(lddata, maxRange);
  - [nidx] = tbot.getOutRangeLidarDataIdx(lddata, maxRange);

- Function naming standardization (read**Sensor**(.), get**Sensor**Data(.)
  - readEncoders(.), getEncodersData(.).
  - readEncodersWithNoise(.), getEncodersDataWithNoise(.).
  - readEncodersTicks(.), getEncodersDataTicks(.).
  - readLidar(.), getLidarData().



See demoGazeboObjects.m

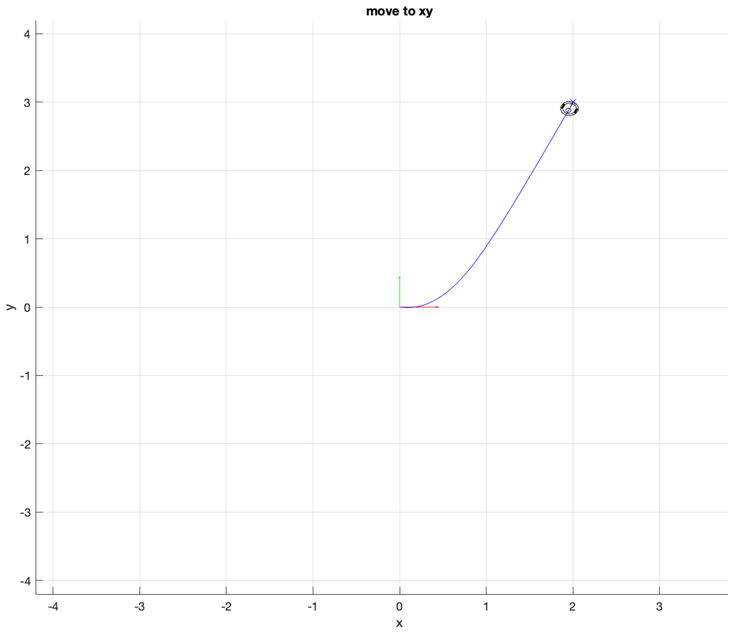
Gazebo control

```
help TurtleBot3.gazeboPlace3DCube
help TurtleBot3.gazeboPlace3DCylinder
help TurtleBot3.gazeboPlace3DCardboardBox
help TurtleBot3.gazeboDeleteAllModels
help TurtleBot3.gazeboPause
help TurtleBot3.gazeboUnPause
help TurtleBot3.gazeboResume
```

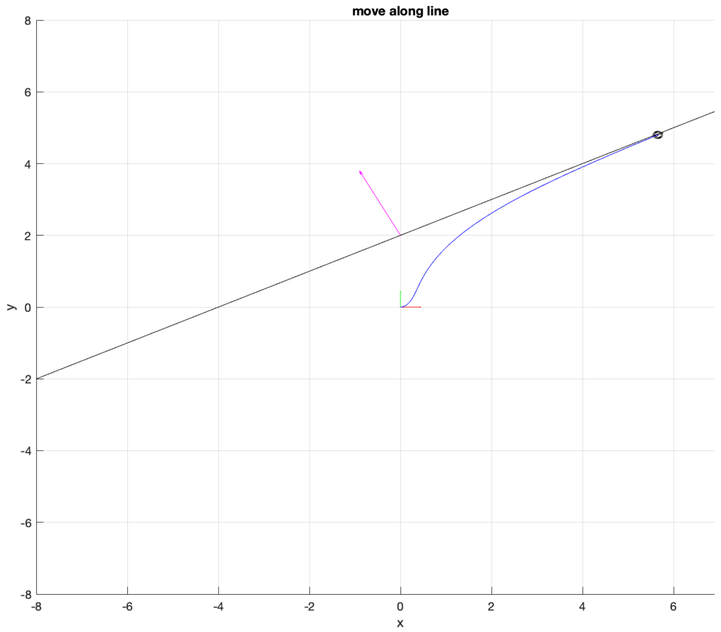
Lab #1

- Move to (x,y) position.
- Move along a line.
- Move to an arbitrary (x,y,θ) pose.

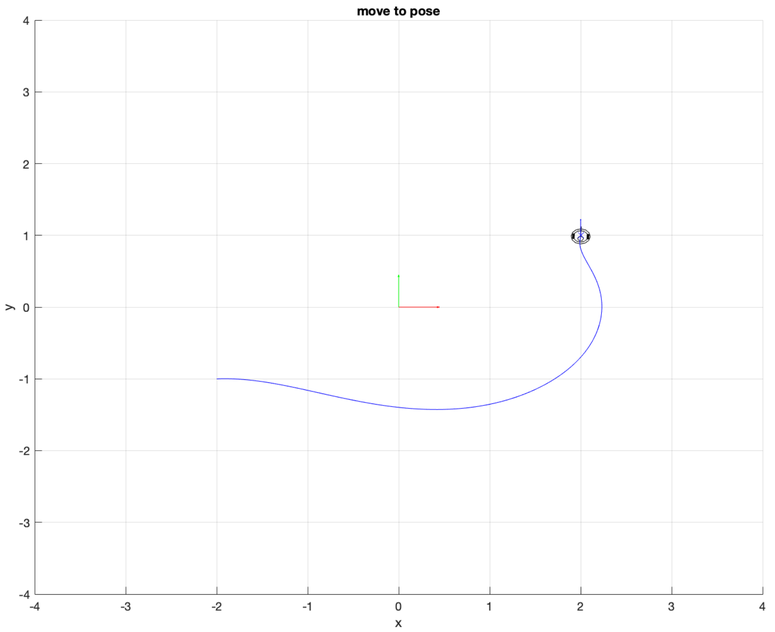
Motion Control



Moving to a point



Moving along a line



Moving to an arbitrary pose

StartUp Code Template

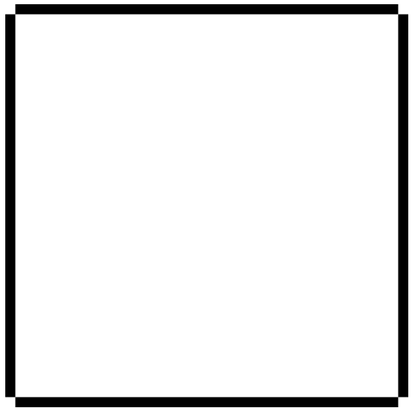
```
% (anonymous function) normalize angle in range [-pi, pi]
normalizeAngle = @(angle) atan2( sin(angle), cos(angle) );

% Max number of iterations
maxIterations = 500;

% Trajectory buffer
trajectory = zeros(maxIterations, 3);

% init ratecontrol obj
r = rateControl(5);     % run at 5Hz (soft constraint)

for it=1:1:maxIterations

    % read TurtleBot pose (x, y, theta and timestamp)
    [x, y, theta, timestamp] = tbot.readPose();

    % normalize theta in range [-pi, pi]
    theta = normalizeAngle(theta);

    % termination criteria
    if( Robot_near_Target_Position ) tbot.stop(); break; end;

    %
    % YOU CODE HERE
    %

    % display
    figure(1); clf; hold on;            % clear figure, hold plots
    drawTurtleBot(x, y, theta);         % draw Robot
    plot(trajectory(1:it,1), trajectory(1:it,2),'b')    % show trajectory
    quiver(0,0,0.5,0,'r')               % draw arrow for x-axis
    quiver(0,0,0,0.5,'g')               % draw arrow for y-axis
    axis([-0.1, 4, -0.1, 4])            % the limits for the current axes [xmin xmax ymin ymax]
    grid on;                            % enable grid
    xlabel('x')                         % axis labels
    ylabel('y')

    % adaptive pause (wait according to rateContol settings)
    waitfor(r);

end

% check statistics of past execution periods
rStats = statistics(r);
```

Lab #2

- 2D + 3D Maps.
- Load a 2D Map from an image file.
- Virtual Force Field (VFF).
- Vector Field Histogram (VFH).

2D Maps

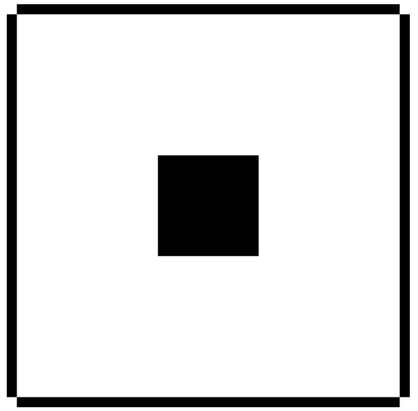
Note: Each map data is stored in image format. MatLab display will look different (see map coordinate systems).



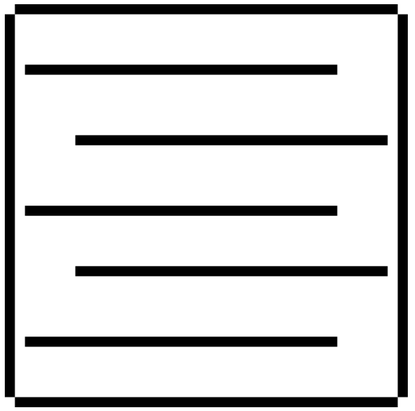
bxmap

csqmap

fmap

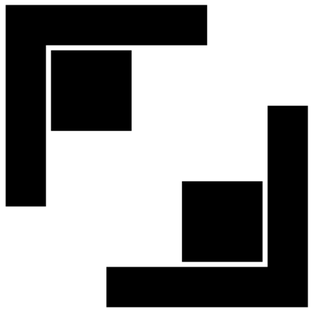


scmap

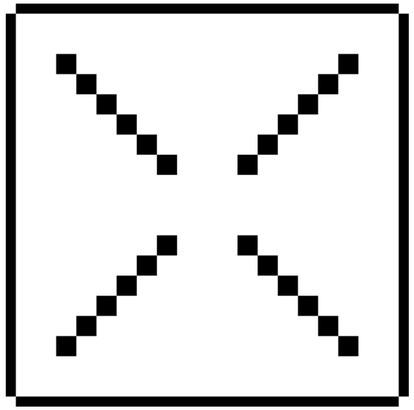


lcmap

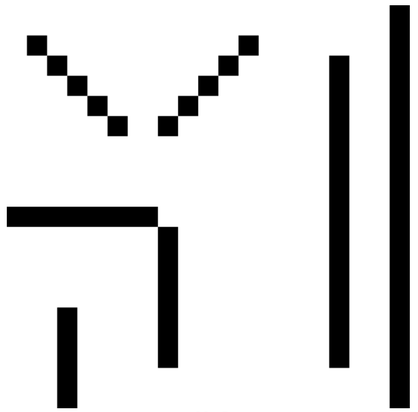
umap



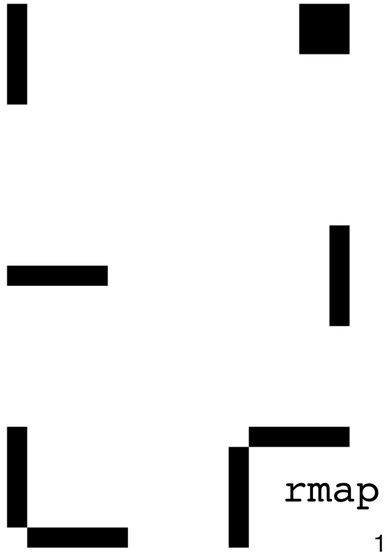
stmap



xmap



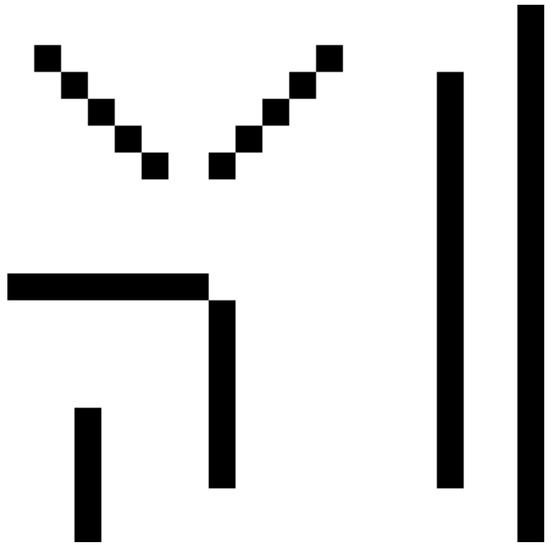
ymap



rmap

Occupation grids

- All maps have a square working area with 4x4 meters *(except **`rmap`** that has rectangular area with 6x4 meters).*
- Several occupation grids (images) are available:
  - **`"map"_grid1.png`** -> each image pixel represents 1x1 cm of working area (**`400x400x3`** image).
  - **`"map"_grid5.png`** -> each image pixel represents 5x5 cm of working area (**`80x80x3`** image).
  - **`"map"_grid10.png`** -> each image pixel represents 10x10 cm of working area (**`40x40x3`** image).
  - Each map has also a .svg image file (vector format).



`ymap (1x1cm grid)`

`ymap (5x5cm grid)`

`ymap (10x10cm grid)`

3D Maps Installation

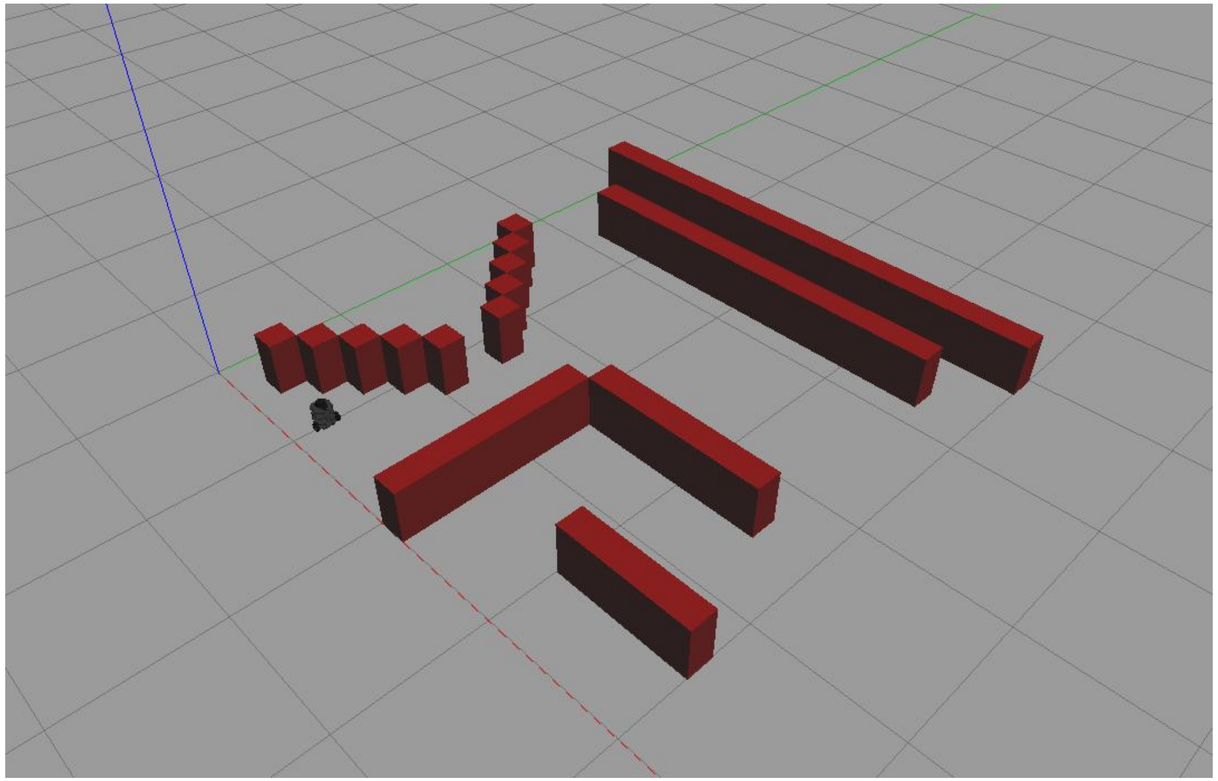
- **Copy the entire "maps" folder to the virtual machine** (to any location):
  - Some virtual machines support simple drag-and-drop functionality.
  - Windows users can use WinSCP to copy data between host and the virtual machine (use sftp connection).
  - MacOs users can use the free Cyberduck App.
  - Linux users just need to write in the location path: sftp://user@192.168.1.xxx

- Open terminal, navigate to the directory location of the maps folder, and run the install script as
  - **>> user@ubuntu:~$ sh install_maps.sh**
    The message "---done---" should appear upon a successful installation.
  - The home folder (/home/user) should hold ten new scripts (start-gazebo-"map".sh).

- Uninstall:
  - >> user@ubuntu:~$ sh uninstall_maps.sh

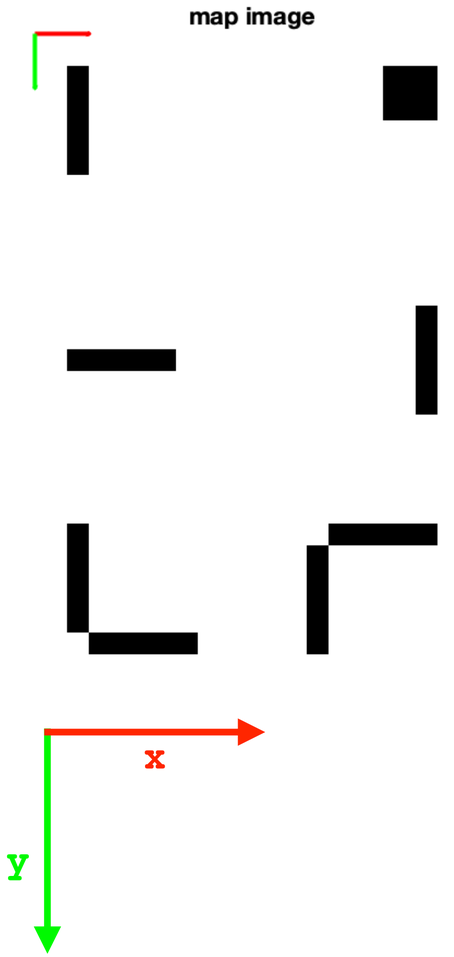
Start Gazebo+ROS with a 3D map

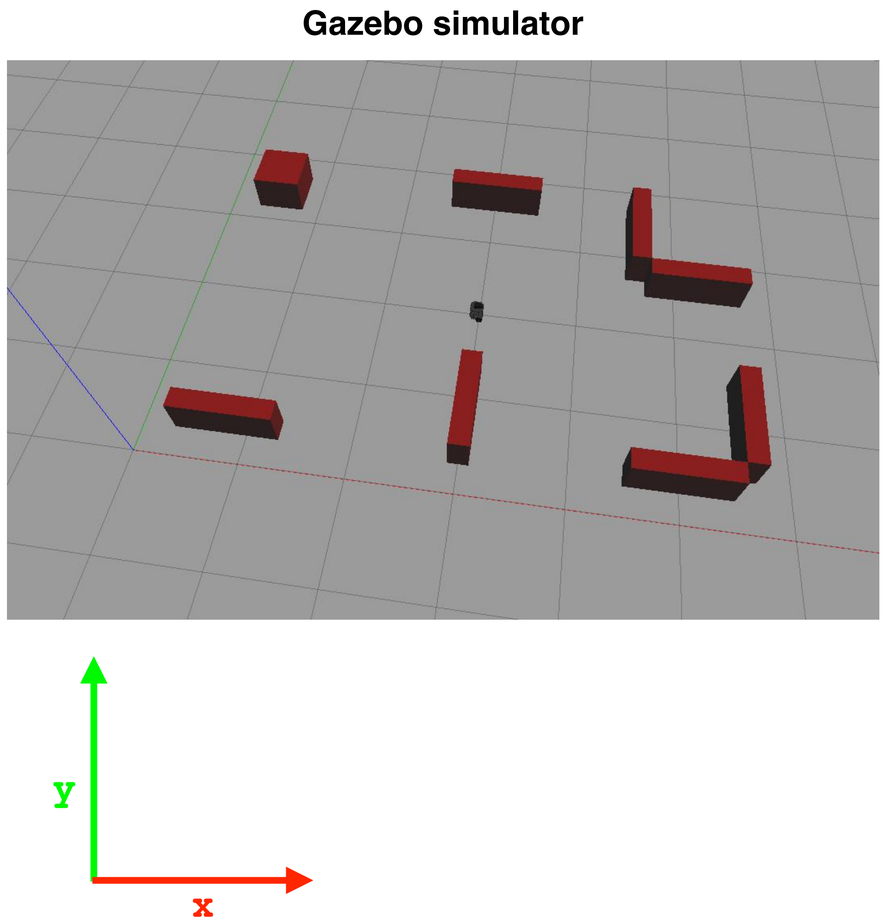
- Run, in the command line, the script associated with the desired map:
- p.e.:
  - `>> user@ubuntu:~$ ./start-gazebo-ymap.sh`

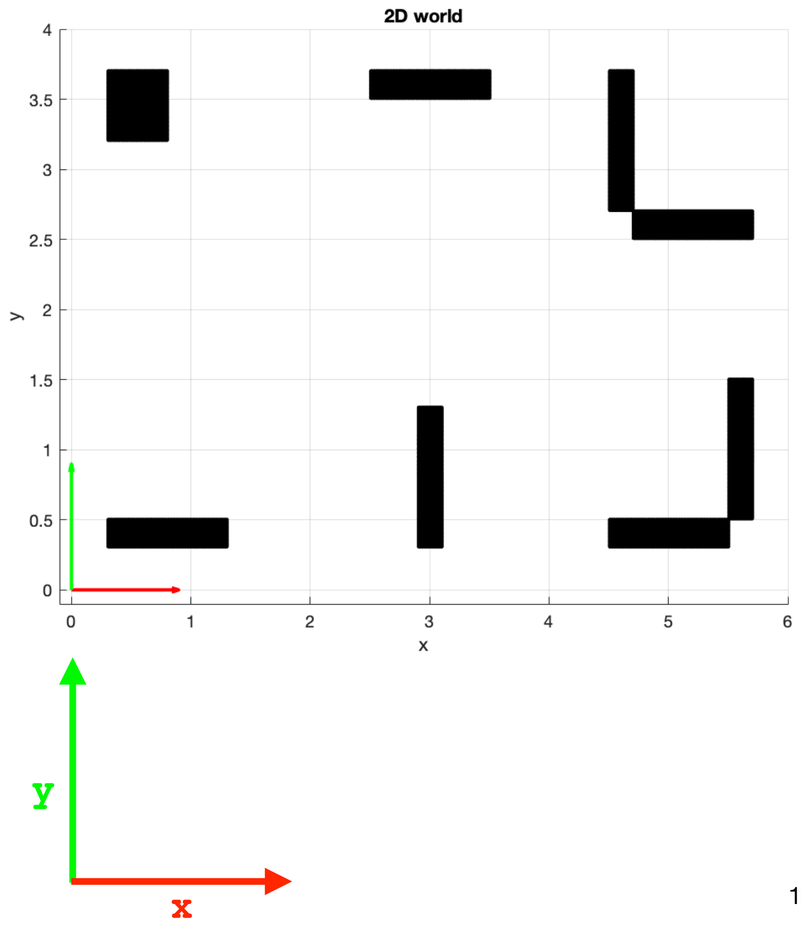


Map Coordinate Systems

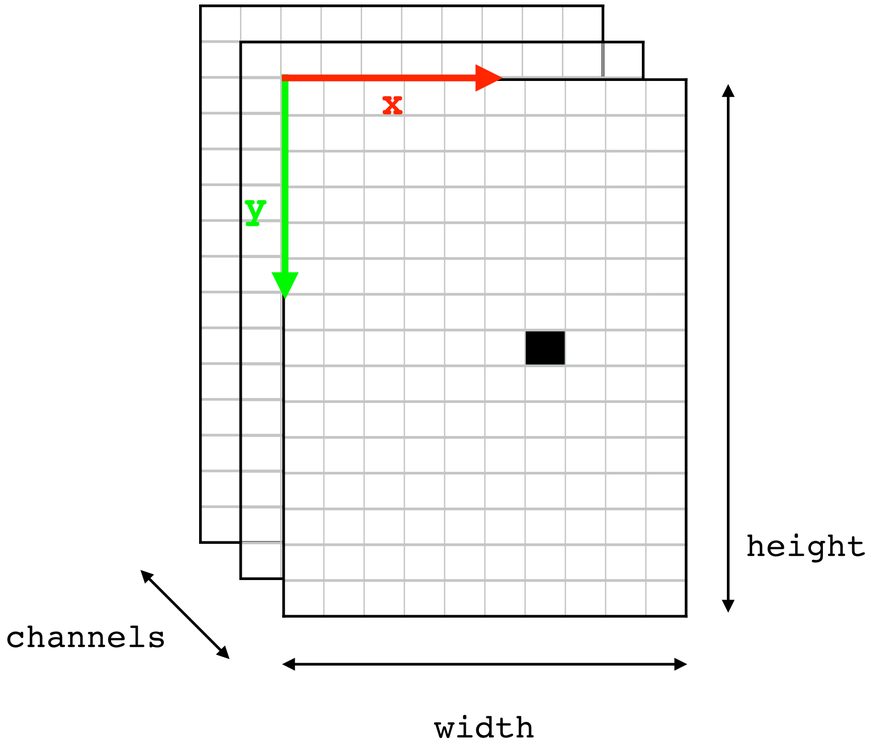
(rmap - rectangular map 6x4m)





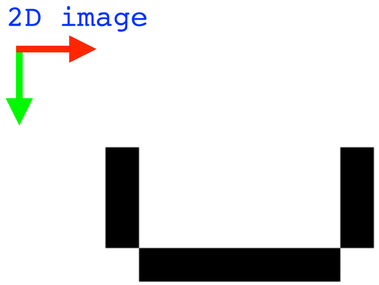


MatLab Image Access

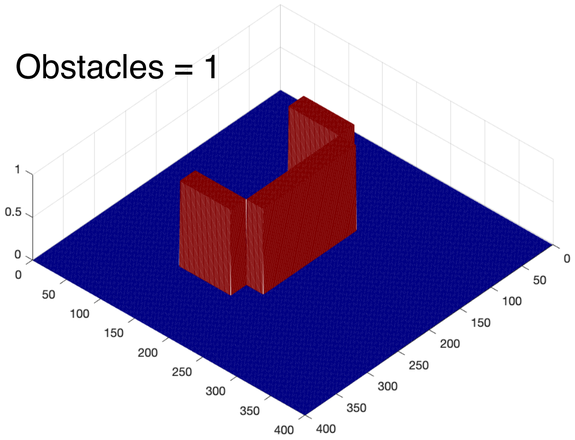



```
% load image
image = imread('map.png');

% get image size
[height, width, channels] = size(image);

% read pixel, p.e. row=100, col=200, ch=1
pxVal = image(row, col, channel);

% show image
figure(1); clf;
imshow(image); hold on;

% plot pixel location (w/ red marker)
plot(col, row,'r+')
```

Load Map data from an Image



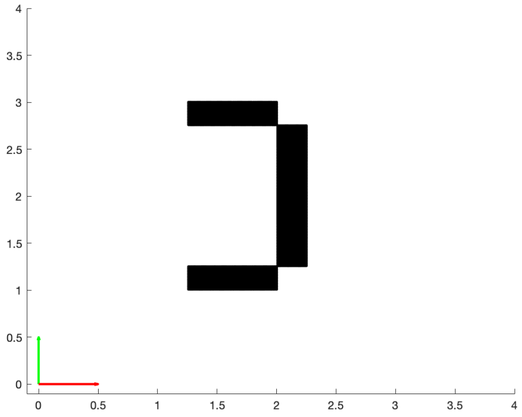
imshow(image)



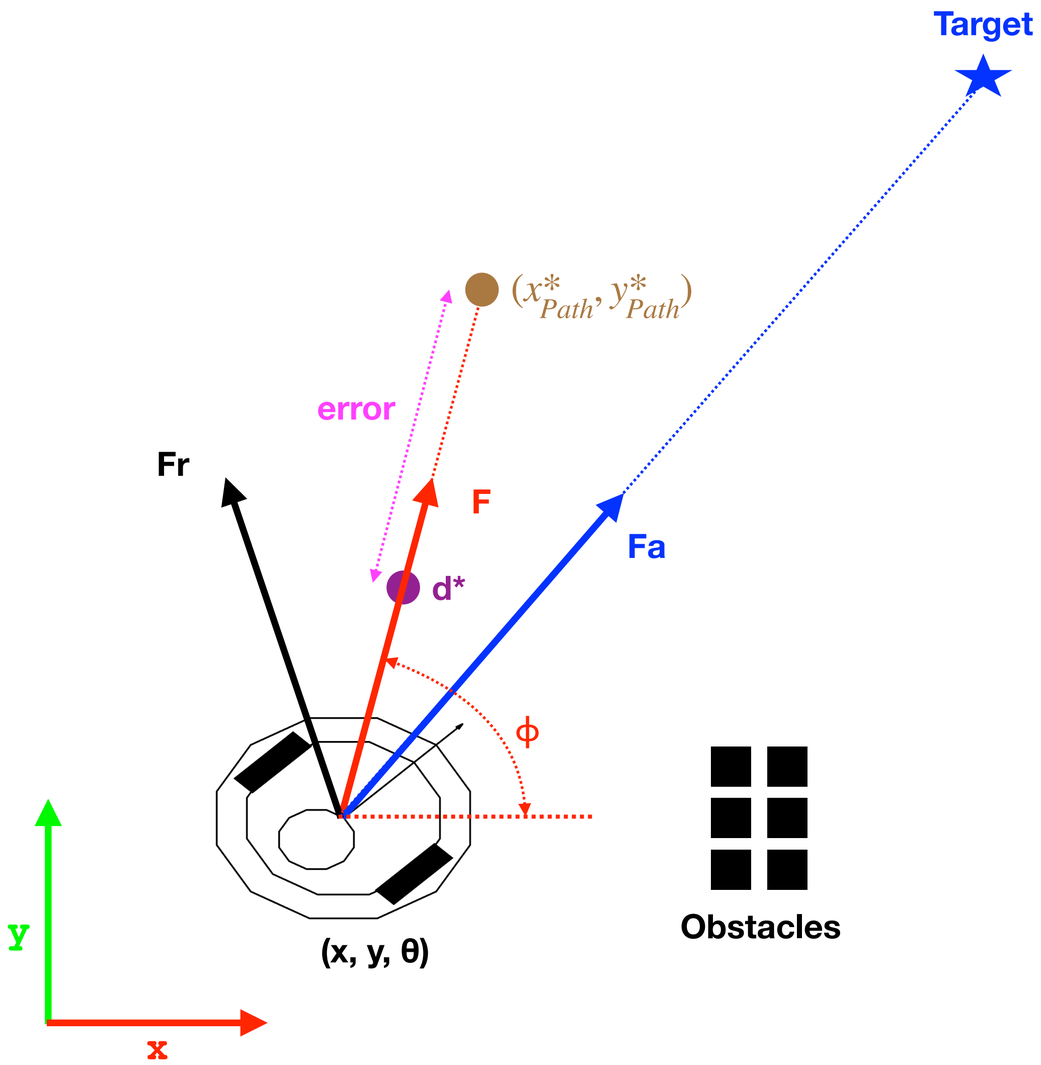
mesh(map)

```
% Load high resolution image map
% image = imread('maps/umap_grid1.png');
image = imread( sprintf('maps/%s_grid1.png', tbot.getGazeboMapName()) );

% define map scale ratio (number of pixels that represent 1 meter)
mscale = 100;    % grid 1x1
%mscale = 20;    % grid 5x5
%mscale = 10;    % grid 10x10

% Convert map into occupation grid format (obstacles = 1)
map = 1 - double( image(:,:,1) ) ./ 255;

% get map size (rows, cols)
[hm, wm] = size(map);

% find (rows, columns) coordinates of obstacles
[rowm, colm] = find( map == 1 );

% display map
figure(1); clf; hold on;
plot( rowm./mscale, colm./mscale, 'k.') % plot obstacles (as black dots)
quiver(0,0,1,0,0.5,'r')                 % draw arrow for x-axis
quiver(0,0,0,1,0.5,'g')                 % draw arrow for y-axis
axis([-0.1, hm/mscale, -0.1, wm/mscale]) % set limits for the current axis
grid on;                                % enable grid
xlabel('x')                             % axis labels
ylabel('y')
title('2D map')
```

MatLab '2D map' plot



Path Following (VFF)




```
% Virtual Force Field
[Fa, Fr] = VFF(currentPose, targetPose, map, searchWindow);
```

$$error = \sqrt{(x^*_{Path} - x)^2 + (y^*_{Path} - y)^2} - d^*$$

$$\phi = \tan^{-1}\left(\frac{y_{Path} - y}{x_{Path} - x}\right)$$

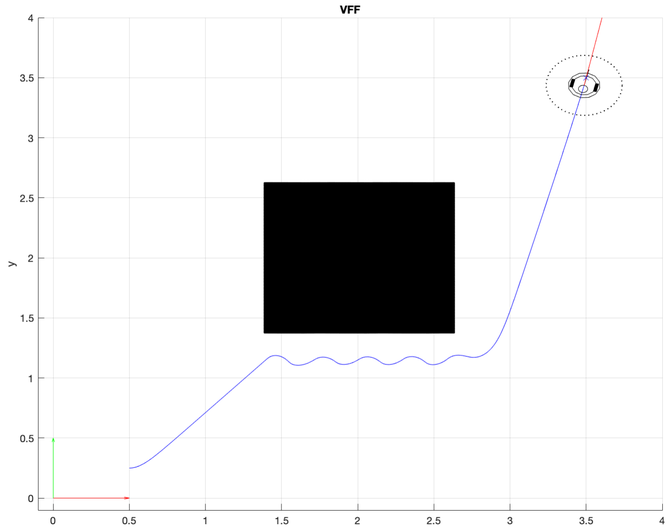
PI Controller (velocity)

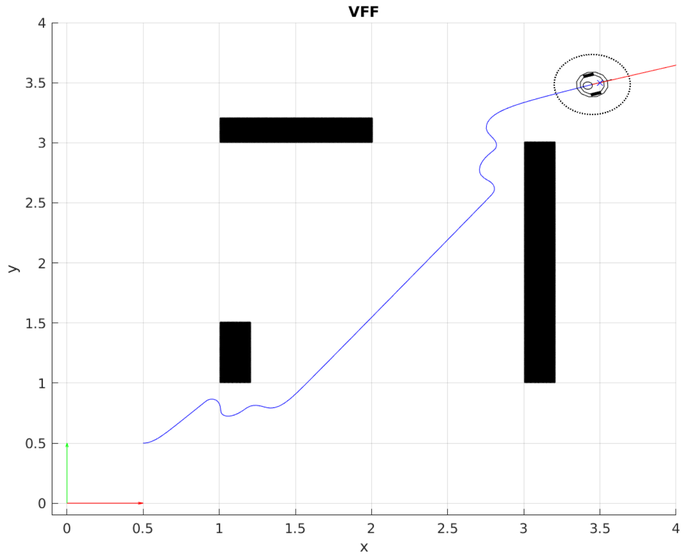
$$v(k) = k_v error(k) + k_i \int error(k) \partial t$$

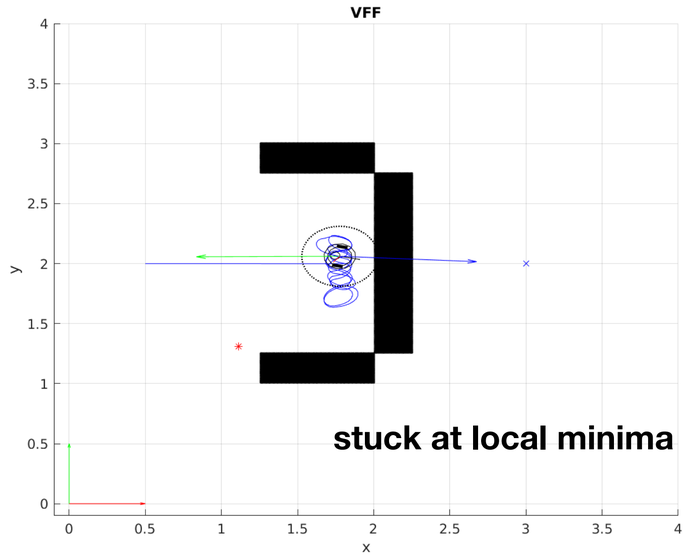
$$error = \sqrt{(x^*_{Path} - x)^2 + (y^*_{Path} - y)^2} - d^*$$

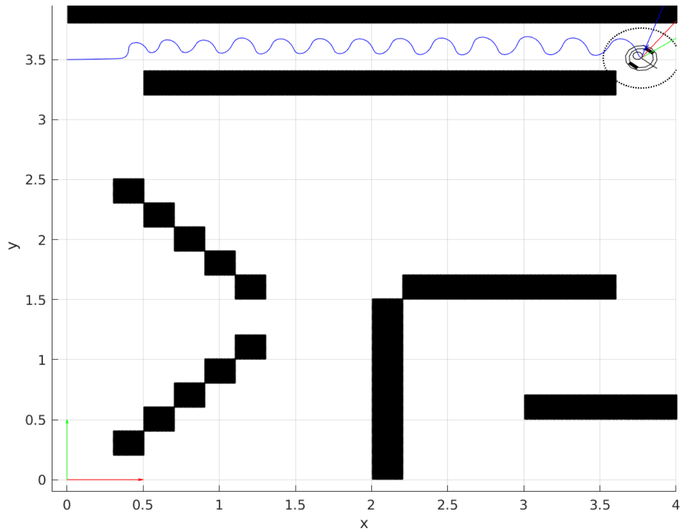
```
% Basic computation of integral error
% IntegralError = IntegralError + Error * dt;

% -> Improved computation of integral error
IntegralError = IntegralError + (Error + previousError) * dt / 2;

% Proportional-Integral (PI) control
v = kv .* Error + ki .* IntegralError;
```

VFF (example trajectories)









VFH Algorithm (high level steps)

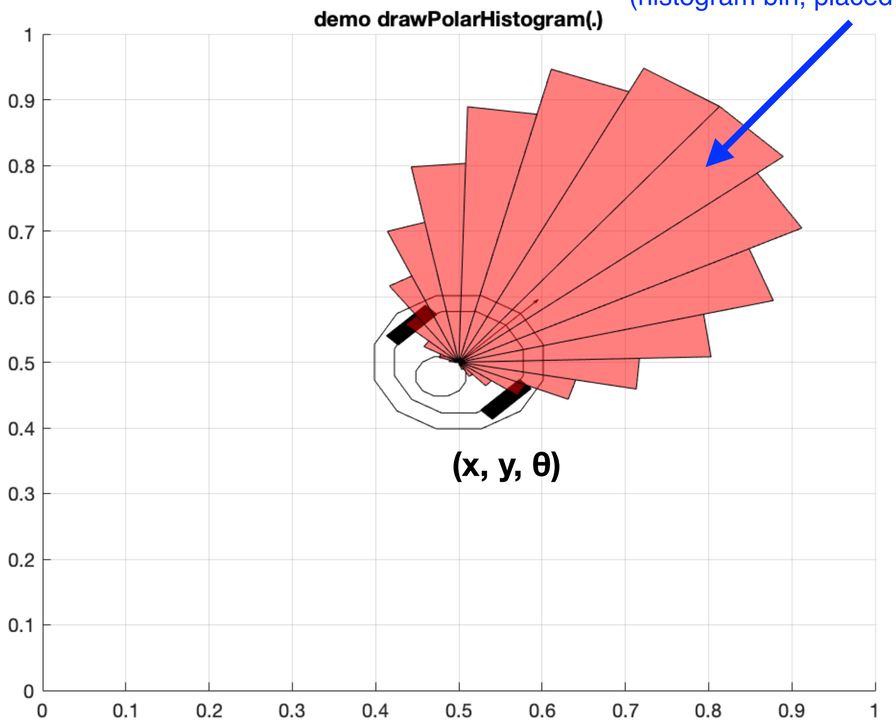
- Define number of sectors (*n*) and an orientation range [-π,π] (or [0,2π]).
- Compute histogram (h):
  - For each obstacle in local search region:
    - compute β, m, and sector k.
    - Add to histogram count: $h(k) = h(k) + m$
- Smooth the histogram (p.e. apply 1D Gaussian filtering).
- Project the target direction into the corresponding histogram bin.
- If there is enough free sectors around the direction of the target: `steerDirection = targetDirection`.
- Find the two nearest “zero crossing” transitions (bin locations) of the histogram (w/ circular wrap around).
  - Ignore histogram counts lower than a small threshold (requires tuning).
- Select the closest bin location (k_n).
- Scan, in the appropriate direction, for s_{max} empty cells.
  - If there is enough empty cells: `steerDirection = (kn +/- smax / 2) α` [wide valley].
  - Otherwise: `steerDirection = (kn1 + kn1) α / 2` [small valley].
  - Stop, if there is no free space.

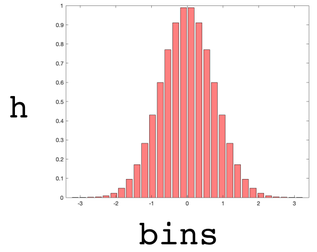
Draw Polar Histogram

The histogram is drawn w.r.t robot orientation (theta).

(for histogram orientation w.r.t. world coordinate system use theta = 0)

(histogram bin, placed at half length: n/2)

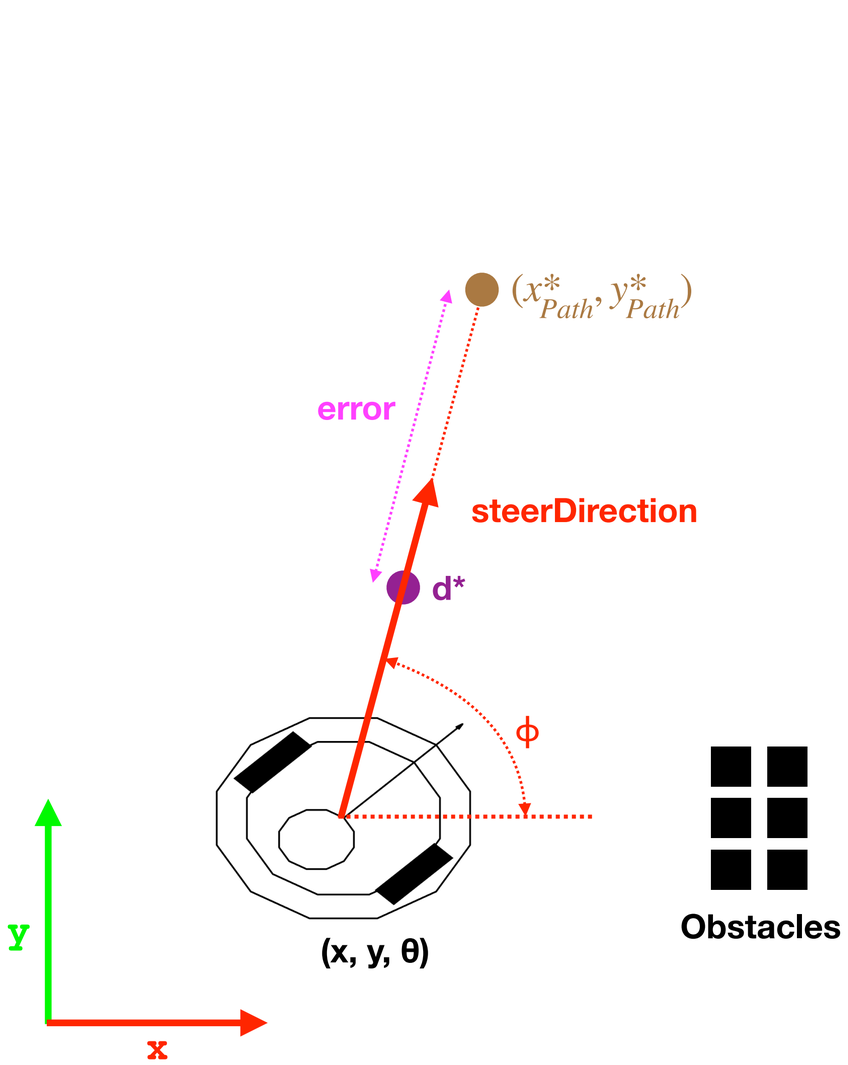





```
% Example usage:

n = 30;                                          % define number of bins
bins = linspace(-pi, pi, n);                     % generate bins in [-pi,pi]

mu = 0; sigma = 0.75;                            % Gaussian parameters
h = exp( - (bins-mu).^2 ./ (2*sigma.^2) );       % generate Gaussian histogram
x = 0.5; y = 0.5; theta = pi/4;                  % define robot pose

figure(1); clf;                                  % init figure & clear graphics
drawTurtleBot(x,y,theta);                        % draw robot
drawPolarHistogram(x, y, theta, h, bins, 0.5, 'r'); % draw polar histogram
axis([0,1,0,1]); grid on;                        % set plot limits
```

`drawPolarHistogram(x, y, theta, h, bins, radius, color);`

Path Following (VFH)




```
% Vector Field Histogram
[steerDirection] = VFH(currentPose, targetPose, map, searchWindow);
```

$$error = \sqrt{(x^*_{Path} - x)^2 + (y^*_{Path} - y)^2} - d^*$$

$$\phi = \tan^{-1}\left(\frac{y_{Path} - y}{x_{Path} - x}\right)$$

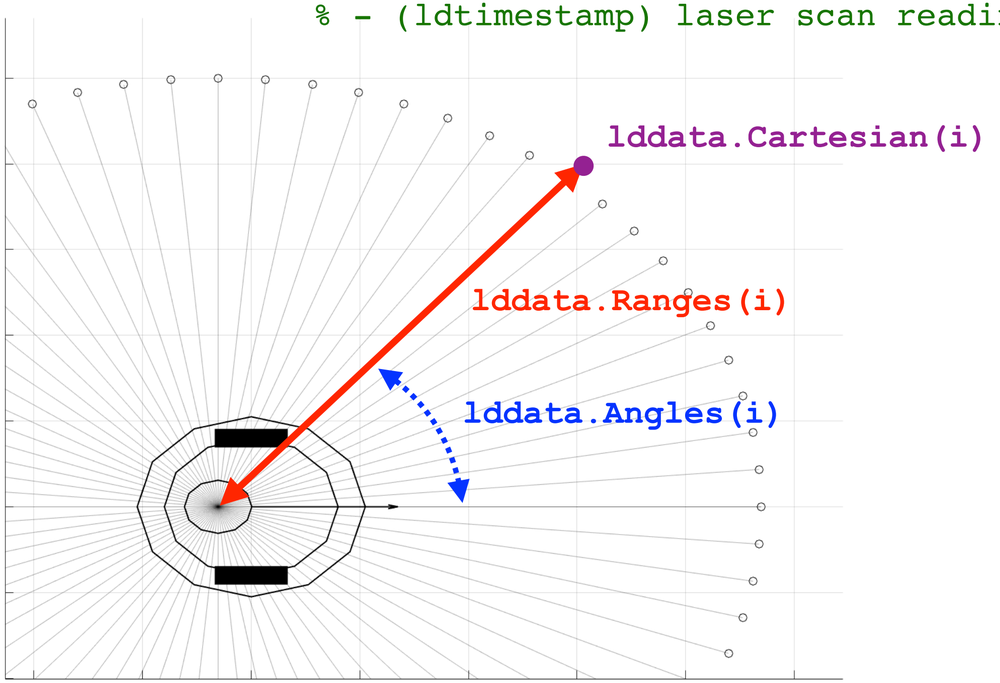
Lab #3

- TurtleBot3 LIDAR functions.
- Convert LIDAR readings into world coordinates.
- Adding Gazebo Objects.
- Map update rules.
- Bresenham line algorithm.
- Map Building Demo.

TurtleBot3 LIDAR functions

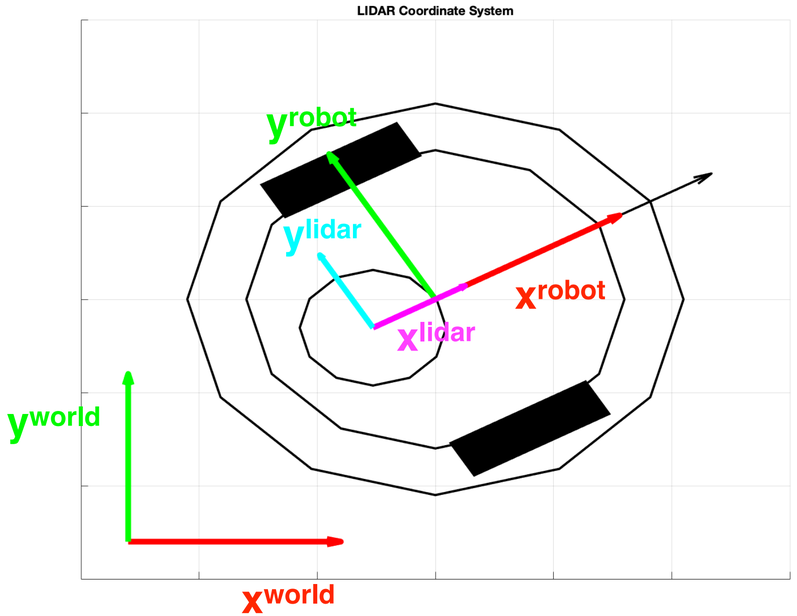
LDS-01 LIDAR
Hitachi-LG Data Storage

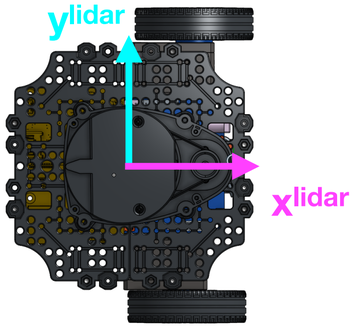
```
% Read lidar data
[scanMsg, lddata, ldtimestamp] = tbot.readLidar();     % or [...] = tbot.getLidarData();

% returns:
% - (scanMsg) lidar obj struct (ROS message)
% - (lddata) lidar data struct
%            - lddata.Ranges - (radial/polar) distance measurement [meters] (360 x 1) vector
%            - lddata.Angles - angular measurement [radians] (360 x 1) vector
%            - lddata.Cartesian - X/Y cartesian data (360 x 2) matrix
% - (ldtimestamp) laser scan reading timestamp (in seconds)
```



- All measurements are given in w.r.t. the LIDAR sensor placed at top of the robot.
- The first array **index** matches the 0º measure: `lddata.Angles(1) = 0.`
- The sensor provides 360 measurements per revolution (~1º angular resolution).
- Angles are expressed in **radians** and distances in **meters**.
- LIDAR Range:
  - Min 12cm
  - Max 3.5m
- The `lsdata` (struct field) arrays could have `'Inf'` (or `zero`) values to represent no laser reflections (representing too near or too far readings).
- The `getInRangeLidarDataIdx(.)` and `getOutRangeLidarDataIdx(.)` functions return the array **`indexes`** of the respective data.

Convert LIDAR readings into global coordinates (“world frame”)





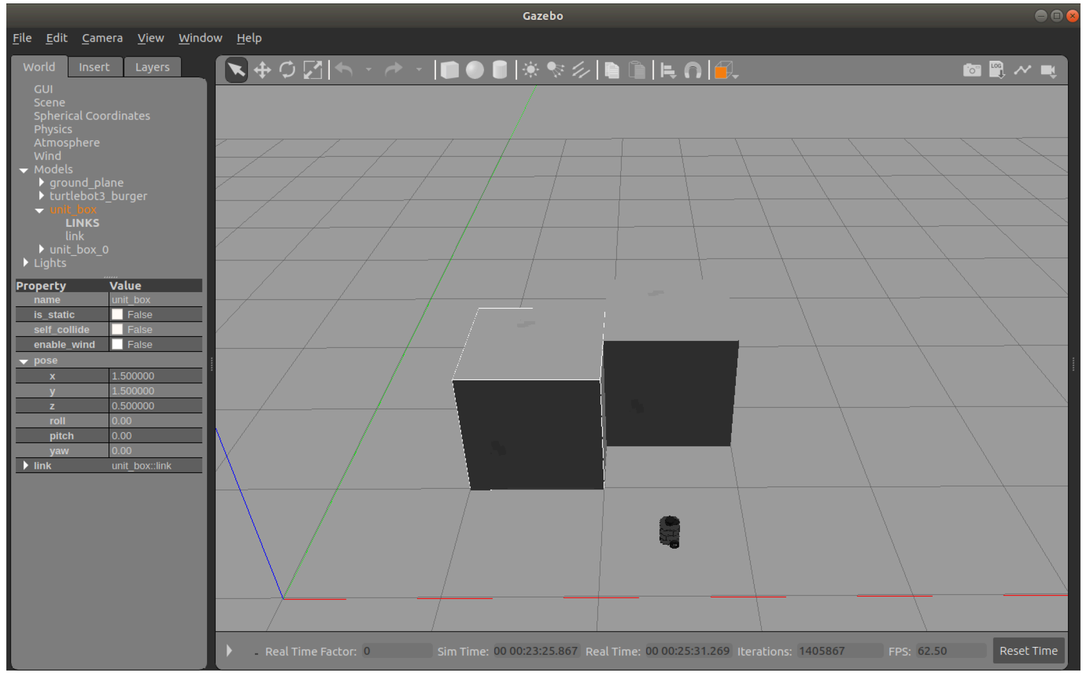
TurtleBot3 top view

$$\begin{pmatrix} x_1^{world} & \dots & x_n^{world} \\ y_1^{world} & \dots & y_n^{world} \\ 1 & \dots & 1 \end{pmatrix}_{3\times n} = \underbrace{\begin{pmatrix} \cos(\theta) & -\sin(\theta) & x - 0.0305\cos(\theta) \\ \sin(\theta) & cos(\theta) & y - 0.0305\sin(\theta) \\ 0 & 0 & 1 \end{pmatrix}_{3\times 3}}_{T(x,y,\theta)} \begin{pmatrix} x_1^{lidar} & \dots & x_n^{lidar} \\ y_1^{lidar} & \dots & y_n^{lidar} \\ 1 & \dots & 1 \end{pmatrix}_{3\times n}$$

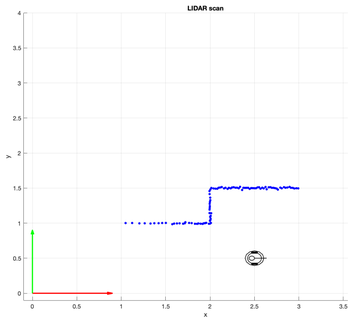
data points at WORLD coordinate system

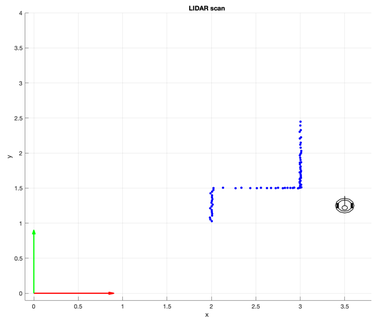
data points at LIDAR coordinate system

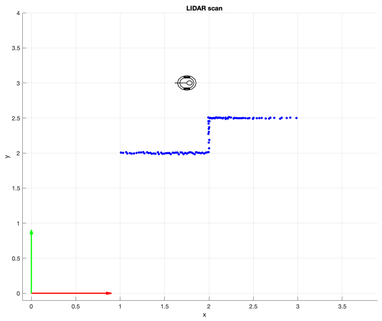
Navigate around Gazebo Objects

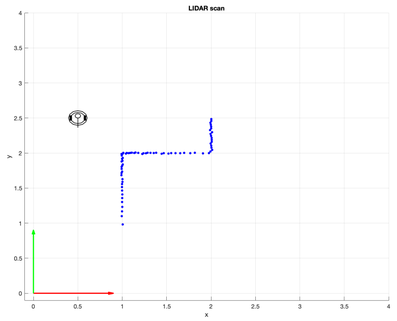


2 cubes placed at (x=1.5, y=1.5) and (x=2.5, y=2), respectively.





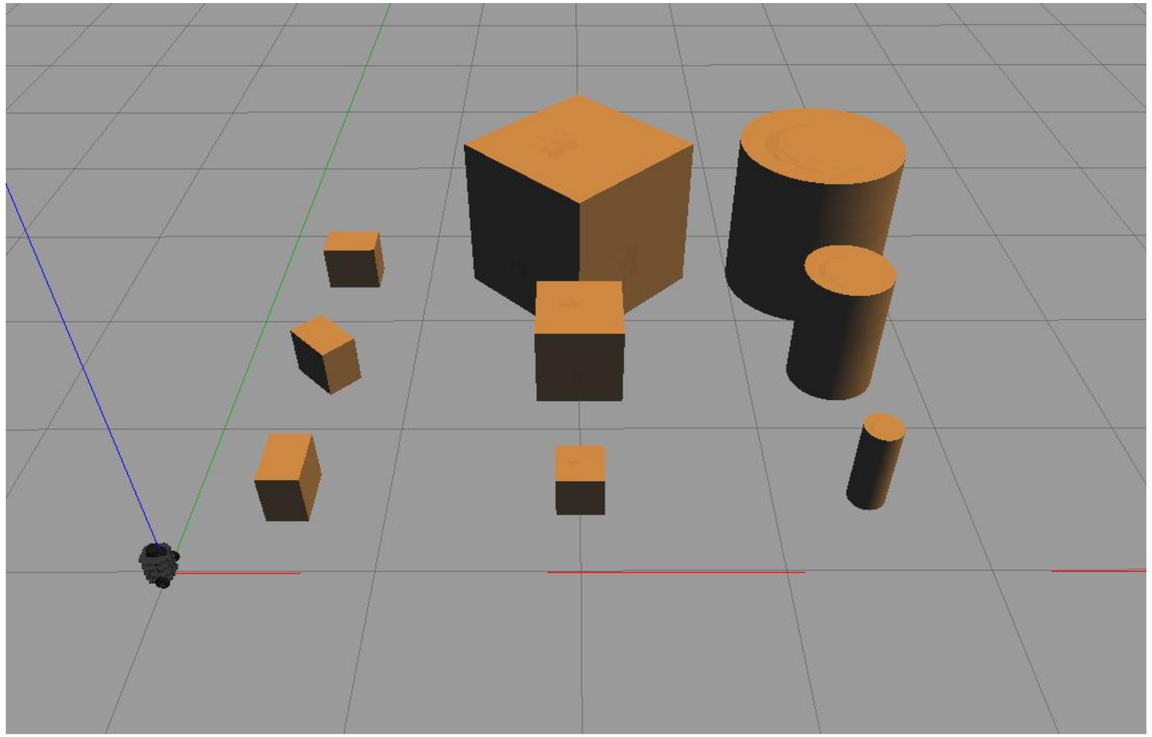




- **Gazebo**
  - Start an empty-world simulation.
  - Place 3D objects (manually):
    - Pause simulation.
    - Add unit cube (edge = 1m).
    - Select the corresponding `unit_box` model at `World->Models` tab.
    - Modify the pose properties.
    - Unpause simulation.
- **MATLAB**
  - Place via-points around objects.
  - Navigate between via-points.
  - Read LIDAR data and convert points into world coordinate frame.

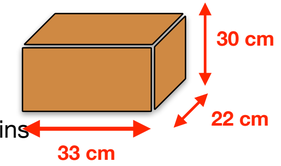
Place Gazebo 3D Objects w/ TurtleBot3 MatLab Interface

- **Requires TurtleBot3 v09c update**



demoGazeboObjects.m

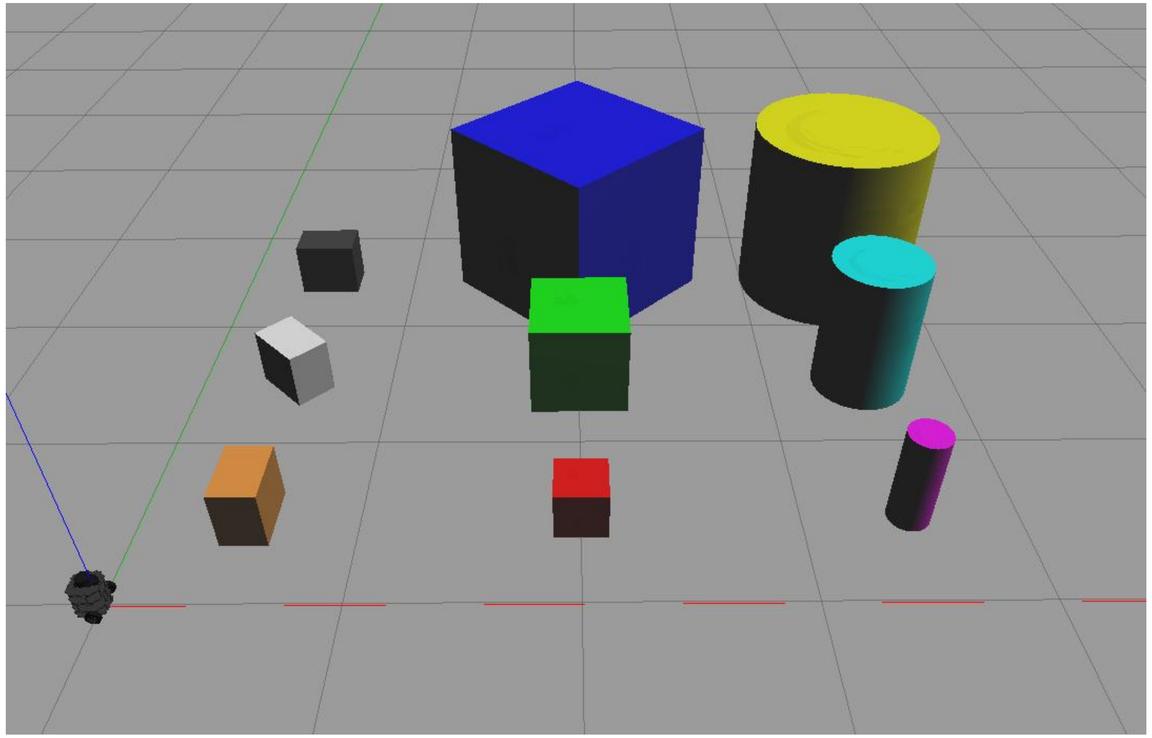
- Cardboard Box model.
  - Size (length x width x height): 33 x 22 x 30 cm




```
% clear memory
clearvars -except tbot;

% start ROS connection and add 3D objects [only if required]
if ( ~exist("tbot") )
    % init by assigning the IPs directly to the TurtleBot constructor
    IP_TURTLEBOT = "192.168.1.xxx";          % virtual machine IP (or robot IP)
    IP_HOST_COMPUTER = “192.168.1.xxx";      % local machine IP
    % Init TurtleBot3 ROS connection & check version
    tbot = TurtleBot3(IP_TURTLEBOT, IP_HOST_COMPUTER);
    if( tbot.getVersion() < 0.92 ) error('TurtleBot3 v09c required'); end

    % ----------------------------------------
    % Place 3D Objects - ONLY ONCE at startup
    % ----------------------------------------
    % Delete all suported 3D objects
    tbot.gazeboDeleteAllModels();

    % Pause Gazebo physics simulation (not required, but faster!)
    tbot.gazeboPause();

    % gazeboPlace3DCardboardBox(x, y, orientation);
    tbot.gazeboPlace3DCardboardBox(0.5, 0.5, 0);
    tbot.gazeboPlace3DCardboardBox(0.5, 1.5, pi/4);
    tbot.gazeboPlace3DCardboardBox(0.5, 2.5, pi/2);

    % gazeboPlace3DCube(x, y, orientation, edgeSize);
    tbot.gazeboPlace3DCube(2, 0.5, 0, 0.25);
    tbot.gazeboPlace3DCube(2, 1.5, 0, 0.5);
    tbot.gazeboPlace3DCube(2, 2.5, pi/4, 1);

    % gazeboPlace3DCylinder(x, y, radius, length);
    tbot.gazeboPlace3DCylinder(3.5, 0.5, 0.1, 0.5);
    tbot.gazeboPlace3DCylinder(3.5, 1.5, 0.25, 0.75);
    tbot.gazeboPlace3DCylinder(3.5, 2.5, 0.5, 1);

    % UnPause/Resume Gazebo physics simulation
    tbot.gazeboUnPause();
end
```

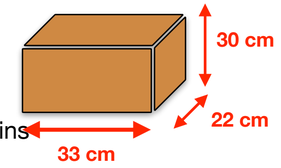
Place Gazebo 3D Objects w/ TurtleBot3 MatLab Interface

- **Requires TurtleBot3 v09d update**



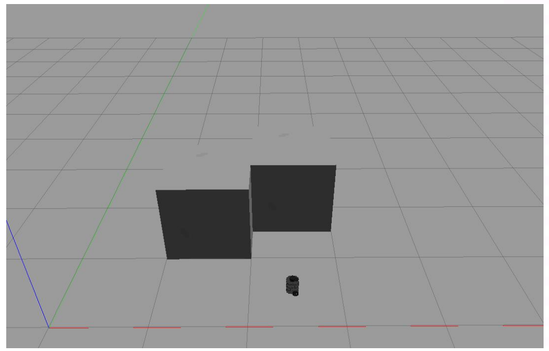
demoGazeboObjects.m

- Cardboard Box model.
  - Size (length x width x height): 33 x 22 x 30 cm

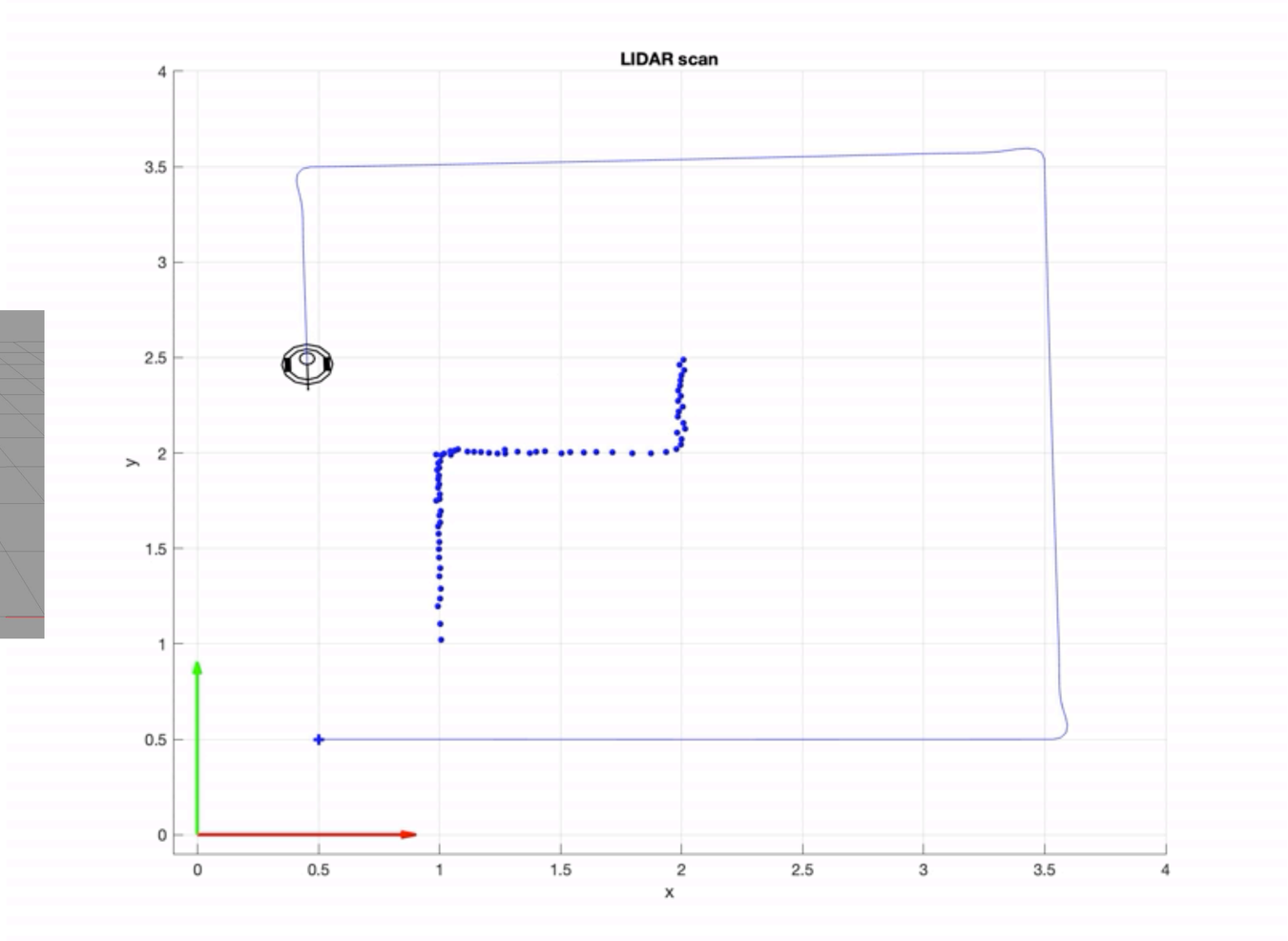



```
% clear memory
clearvars -except tbot;

% start ROS connection and add 3D objects [only if required]
if ( ~exist("tbot") )
    % init by assigning the IPs directly to the TurtleBot constructor
    IP_TURTLEBOT = "192.168.1.xxx";          % virtual machine IP (or robot IP)
    IP_HOST_COMPUTER = “192.168.1.xxx";      % local machine IP
    % Init TurtleBot3 ROS connection & check version
    tbot = TurtleBot3(IP_TURTLEBOT, IP_HOST_COMPUTER);
    if( tbot.getVersion() < 0.93 ) error('TurtleBot3 v09d required'); end

    % ----------------------------------------
    % Place 3D Objects - ONLY ONCE at startup
    % ----------------------------------------
    % Delete all suported 3D objects
    tbot.gazeboDeleteAllModels();

    % Pause Gazebo physics simulation (not required, but faster!)
    tbot.gazeboPause();

    % gazeboPlace3DCardboardBox(x, y, orientation, color);    # custom [r,g,b] color
    tbot.gazeboPlace3DCardboardBox(0.5, 0.5, 0);                    % default color
    tbot.gazeboPlace3DCardboardBox(0.5, 1.5, pi/4, 'w');            % white box
    tbot.gazeboPlace3DCardboardBox(0.5, 2.5, pi/2, [0.3, 0.3, 0.3]); % dark grey box

    % gazeboPlace3DCube(x, y, orientation, edgeSize, color);
    tbot.gazeboPlace3DCube(2, 0.5, 0, 0.25, 'r');
    tbot.gazeboPlace3DCube(2, 1.5, 0, 0.5, 'g');
    tbot.gazeboPlace3DCube(2, 2.5, pi/4, 1, 'b');

    % gazeboPlace3DCylinder(x, y, radius, length, color);
    tbot.gazeboPlace3DCylinder(3.5, 0.5, 0.1, 0.5, 'm');
    tbot.gazeboPlace3DCylinder(3.5, 1.5, 0.25, 0.75, 'c');
    tbot.gazeboPlace3DCylinder(3.5, 2.5, 0.5, 1, 'y');

    % UnPause/Resume Gazebo physics simulation
    tbot.gazeboResume();
end
```

LIDAR Example - Navigate around 2 Cubes



Gazebo Objects



Navigate w/ Map Building

```
% init map (probability of each cell being obstacle = 50%)
map = 0.5 .* ones(h, w);

% Read Odometry data
[x, y, theta, timestamp] = tbot.readPose();

% Read LIDAR data
[scanMsg, lddata, ldtimestamp] = tbot.readLidar();

% Convert LIDAR data into World Coordinates
# ...

% convert probability into log odds
map = log( map ./ (1 - map ) );

% Map Building Process
[map] = MapUpdate(map, robotPose, LidarData, ...);

% convert log odds back to probability
map = 1 - 1 ./ (1 + exp(map) );

% Run Navigation Algorithm
# VFF(map, ...) or VFH(map, ...);
```

Map Update (w/ Inverse Sensor Model)

Algorithm inverse_range_sensor_model(i, x_t, z_t)**:**

Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$

$r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$

$\phi = \text{atan2}(y_i - y, x_i - x) - \theta$

$k = \text{argmin}_j \, |\phi - \theta_{j,\text{sens}}|$

if $r > \min(z_{\max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{k,\text{sens}}| > \beta/2$ then

  return l_0

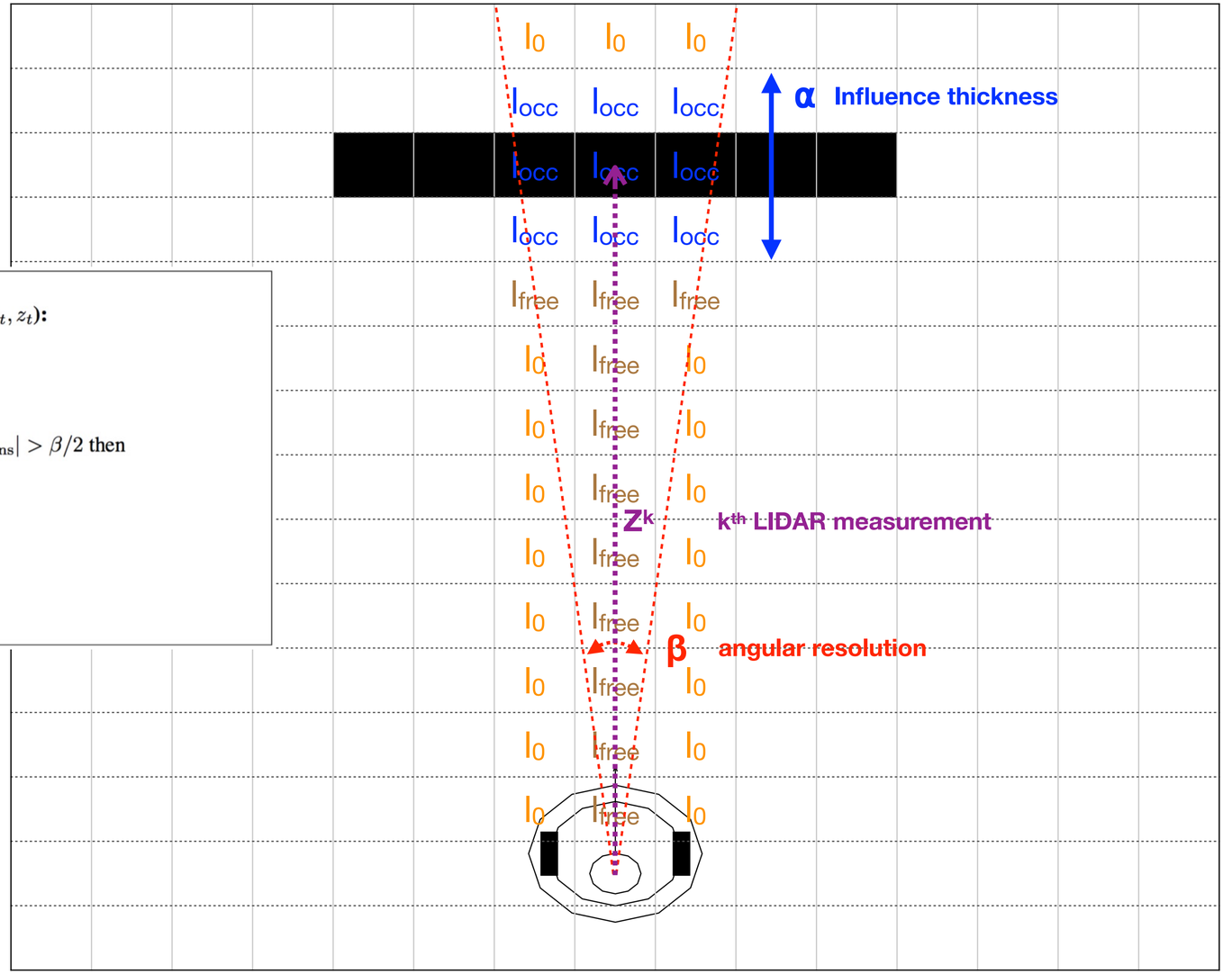
if $z_t^k < z_{\max}$ and $|r - z_t^k \quad | < \alpha/2$

  return l_{occ}

if $r \leq z_t^k - \alpha/2$

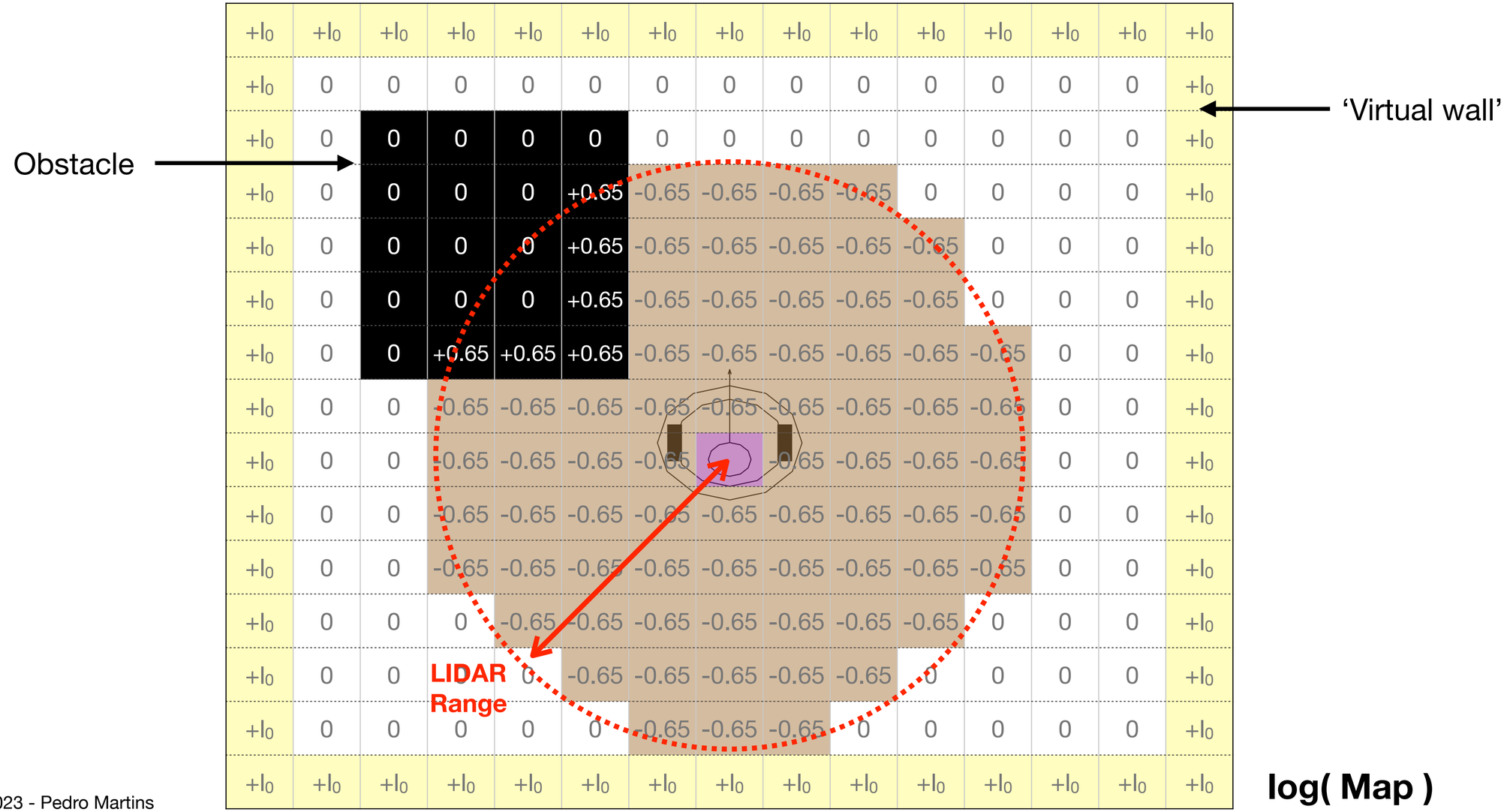
  return l_{free}

endif



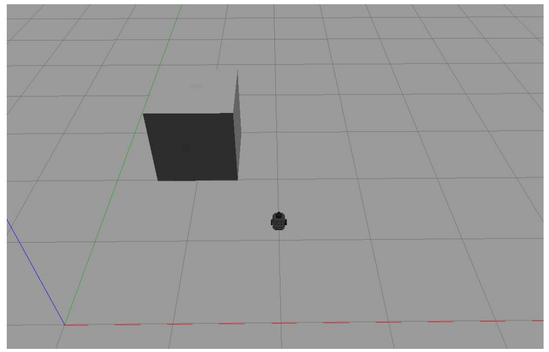
single LIDAR measurement

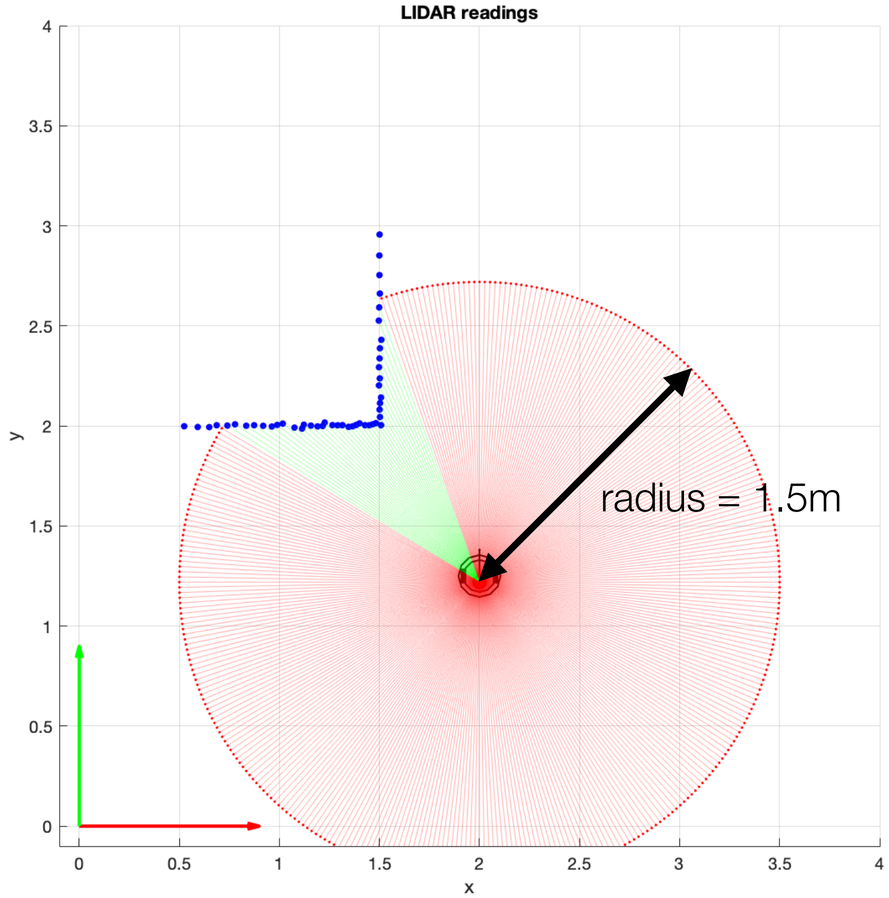
Map Update (simplified model)

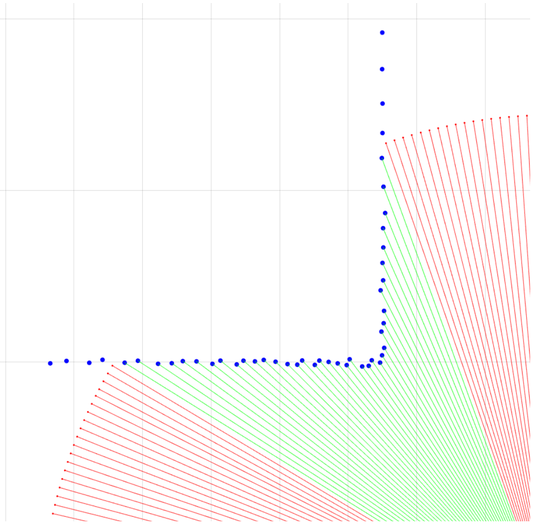


In Range / Out of Range - LIDAR Readings

MinRange (12cm) < Valid Measure < MaxRange (3.5m)



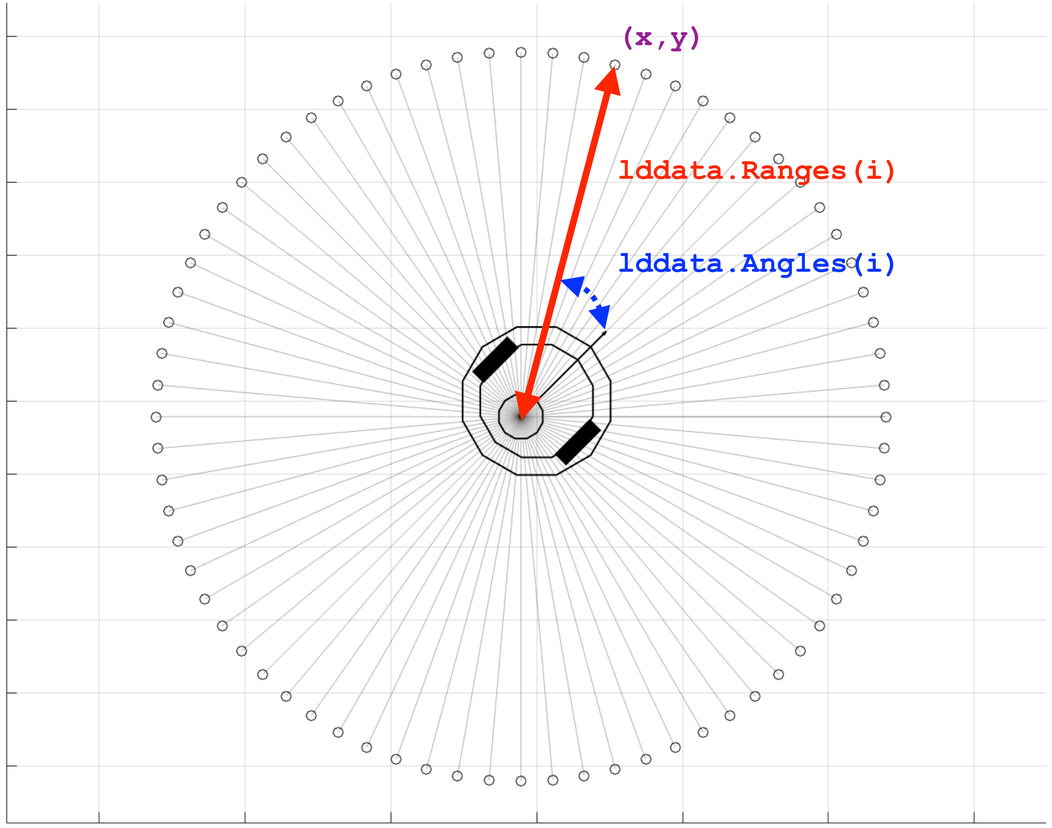




Detailed view

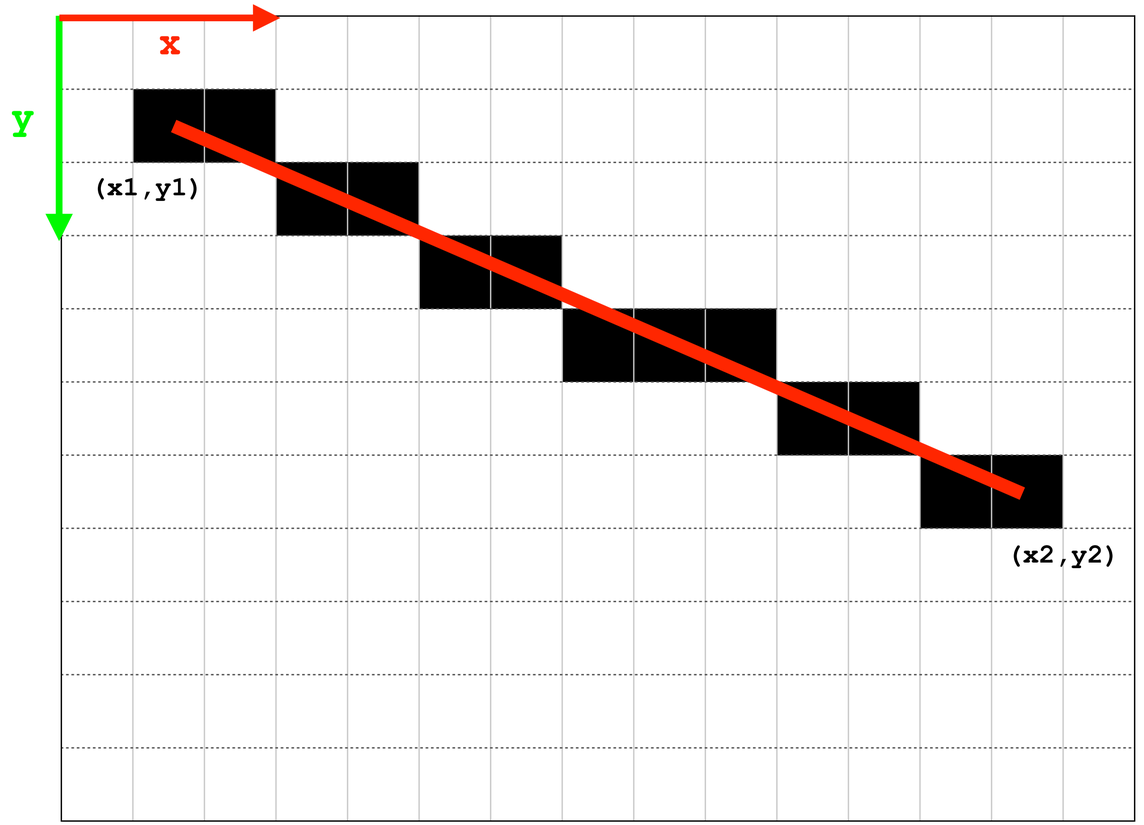
```
% Get in range LIDAR data indexes
[vidx] = tbot.getInRangeLidarDataIdx(lddata, maxRange);

% Get the out of range LIDAR data indexes
[nidx] = tbot.getOutRangeLidarDataIdx(lddata, maxRange);
```

LIDAR Readings

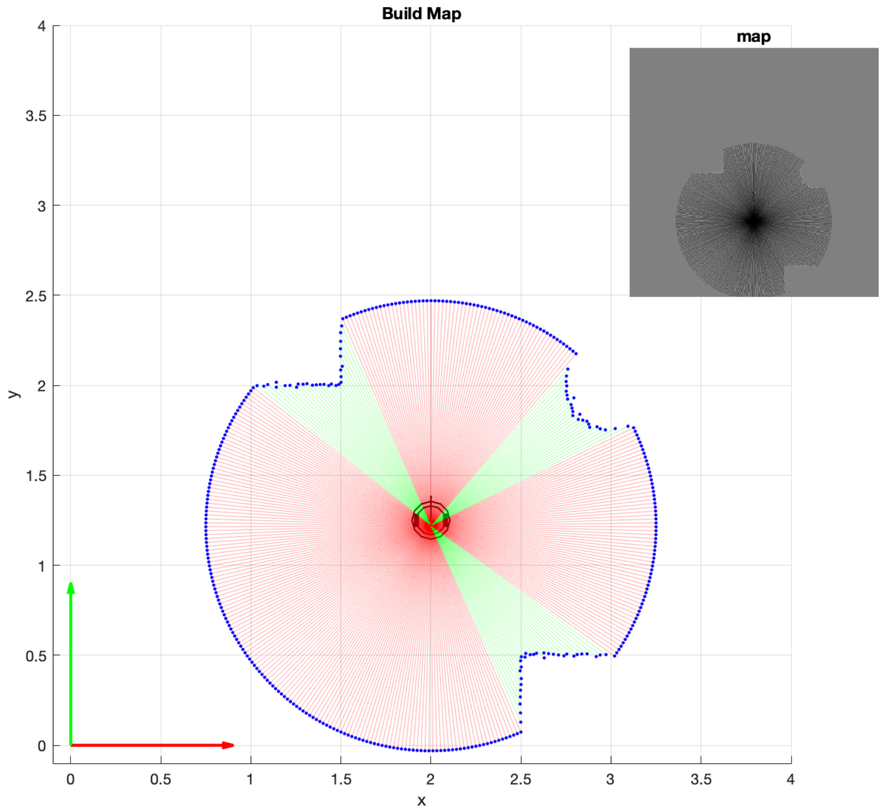


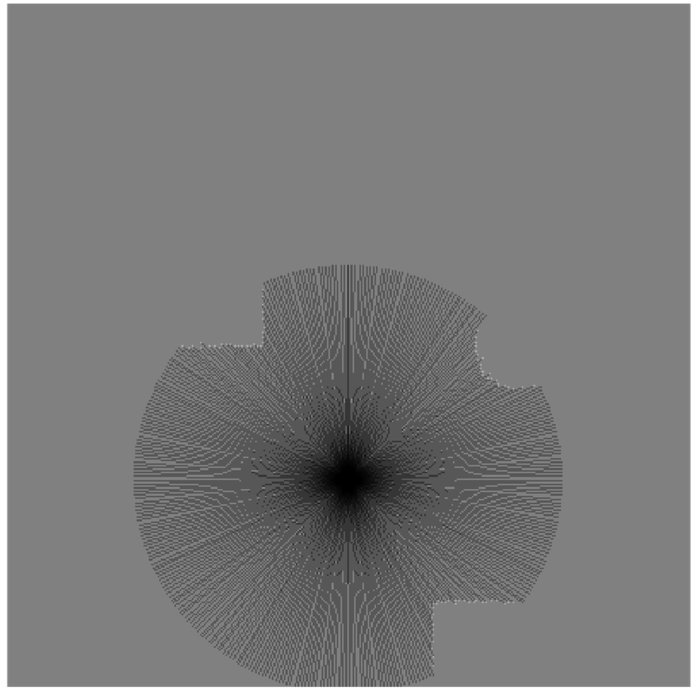
Bresenham Line Algorithm

```
% Bresenham line algorithm
[x, y] = bresenham( x1, y1, x2, y2);

% (x1,y1) - start position
% (x2,y2) - end position
% return (x,y) - vectors w/ line coordinates from (x1,y1) to (x2,y2)
```

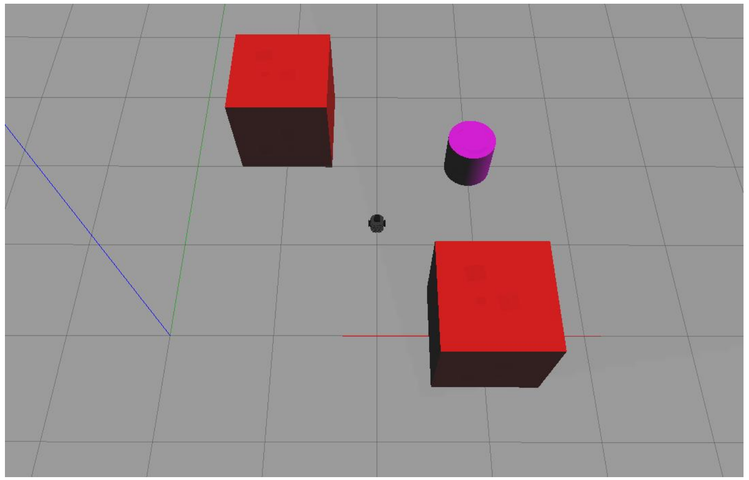


Map Building (single update)



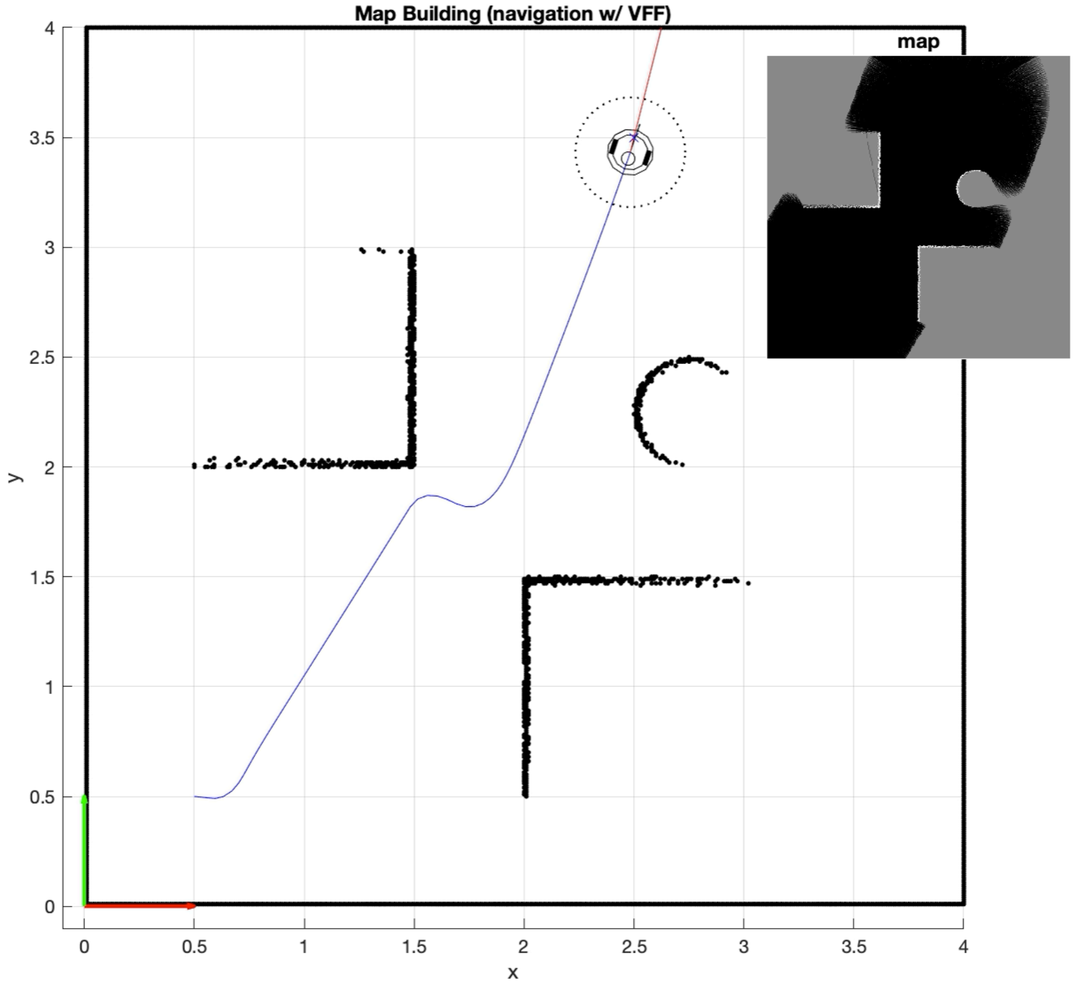


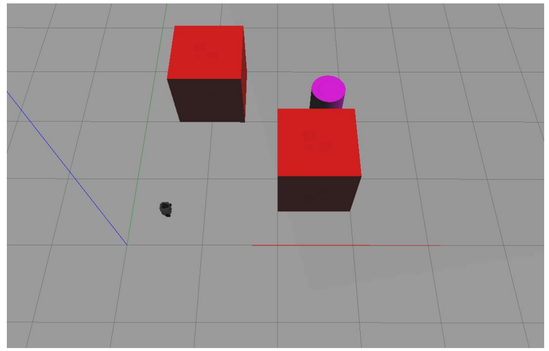
Detailed view



3D Map

Map Building (demo)





3D Map

Lab #4

- Differential Drive Kinematics Model
- TurtleBot3 Encoders data functions.
- Demo EKF localization.
- Extended Kalman Filter eqs.
- Kalman Prediction
- Kalman Correction/Update
- Observation - Prediction Correspondences.
- StartUp Code.
- Gradient Computation (w/ Symbolic MatLab Toolbox).

Differential Drive Kinematics Model

$$f(p,u) = \begin{pmatrix} x + D\cos(\theta + \frac{\Delta\theta}{2}) \\ y + D\sin(\theta + \frac{\Delta\theta}{2}) \\ \theta + \Delta\theta \end{pmatrix}$$

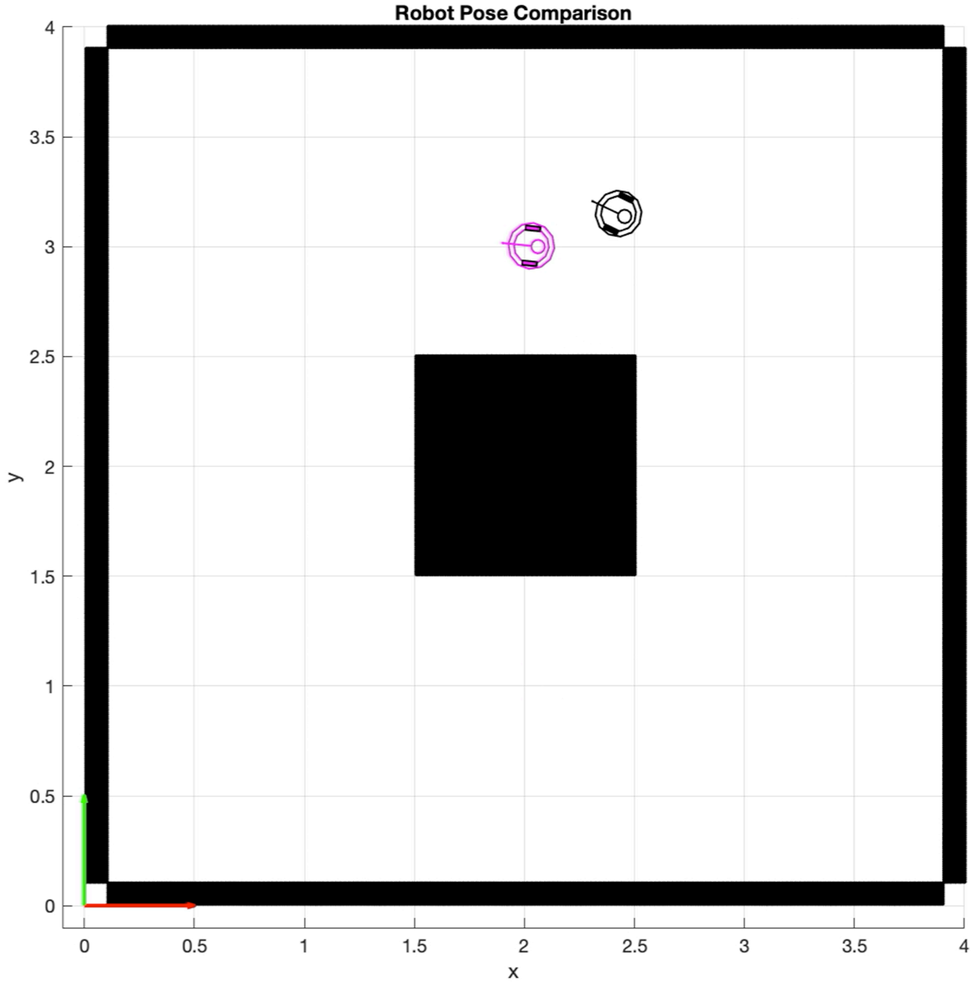
$$p = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

$$u = \begin{pmatrix} D_r \\ D_l \end{pmatrix}$$

$$D = \frac{Dr + Dl}{2}$$

$$\Delta\theta = \frac{Dr - Dl}{b}$$

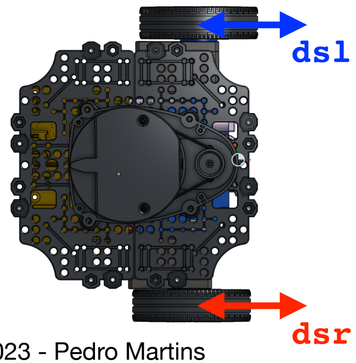
- D_r - Displacement of right wheel
- D_l - Displacement of left wheel
- b - baseline (separation between wheels)



- Black Robot (baseline = 16cm)
- Magenta Robot (baseline = 18cm)

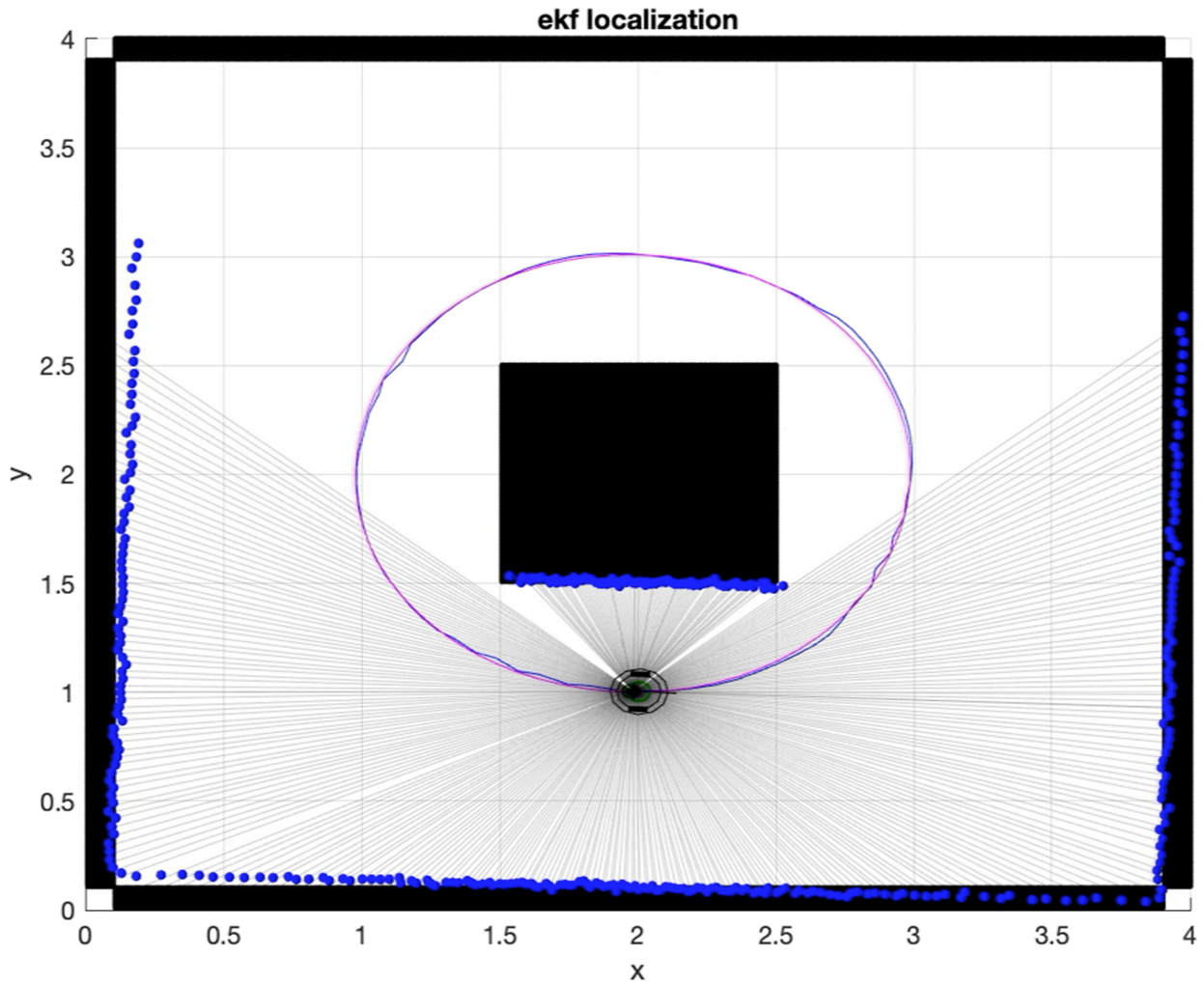
TurtleBot3 Encoders data functions

```
% Initialize the encoders module (required at startup).
tbot.initEncoders();

% Get (simulated) encoders data.
[dsr, dsl, pose2D, timestamp] = tbot.readEncoders();

% returns:
%  - (dsr) incremental motion of the right wheel [in meters].
%  - (dsl) incremental motion of the left wheel [in meters].
%  - (pose2D) [x,y,theta] 2D pose given by the TurtleBot's odometry model.
%  - (timestamp) data reading timestamp [seconds].

% Get (simulated) Encoders Data + Gaussian Noise
[dsr, dsl, pose2D, timestamp] = tbot.readEncodersWithNoise(noise_std);

% input:
%  - (noise_std) 2 x 1 vector w/ noise standard deviations of dsr and dsl [in meters].
%  - if noise_std is a scalar value, the same amount of noise is applied in both readings (dsl and dsl).
```



- Provides simulated data (check source code).
- Must be initialized by `initEncoders()` function.
- The `readEncoders()`, `readEncodersWithNoise()` and `readPose()` can't be used together in a loop.

EKF Localization Demo



Extended Kalman Filter Eqs.

state vector

$$p_k = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

$$\left(p_k, \Sigma_k\right) = EKF\left(p_{k-1}, \Sigma_{k-1}, u_k, z_k\right)$$

(3 x 1) (3 x 3)

previous state vector

previous state covariance

control input

measurement / observation

Prediction

$$\bar{p}_k = f(p_{k-1}, u_k)$$

process noise covariance

$$\bar{\Sigma}_k = \nabla f_p \, \Sigma_{k-1} \nabla f_p^T + \nabla f_u \, \Sigma_u \nabla f_u^T + Q_k$$

Correction / Update

Kalman gain

noise covariance

$$K = \bar{\Sigma}_k \nabla h^T \underbrace{(\nabla h \, \bar{\Sigma}_k \, \nabla h^T + R_k \,)}_{S}{}^{-1}$$

S → inovation covariance

state vector

$$p_k = \bar{p}_k + K \underbrace{\left(z_k - h(\bar{p}_k)\right)}_{v}$$

v → inovation

state covariance

$$\Sigma_k = \left(I_3 - K \, \nabla h\right) \bar{\Sigma}_k$$

- Note:
  - Some gradient were derived in the main lecture.
  - The MatLab Symbolic Toolbox the toolbox can be used to calculate the missing gradients.

EKF - Prediction

predicted state vector

$$\bar{p}_k = f(p_{k-1}, u_k)$$

(3 x 1) kinematic model

predicted state covariance

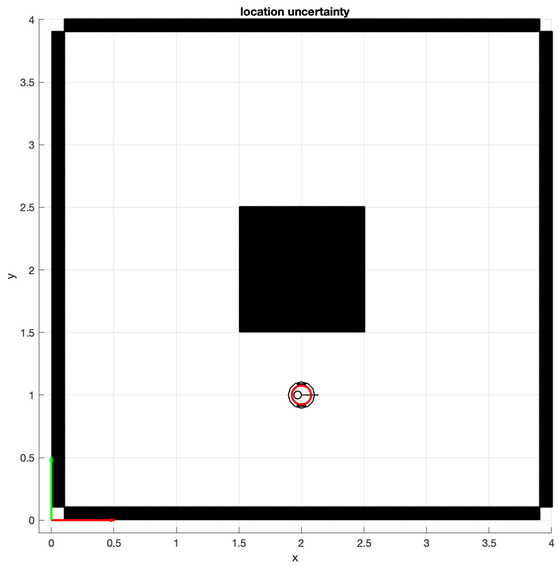
$$\bar{\Sigma}_k = \nabla f_p \; \Sigma_{k-1} \nabla f_p^T + \nabla f_u \; \Sigma_u \nabla f_u^T + Q_k$$

(3 x 3) (3 x 3) (3 x 3) (3 x 3) (3 x 2) (2 x 2) (2 x 3) (3 x 3)

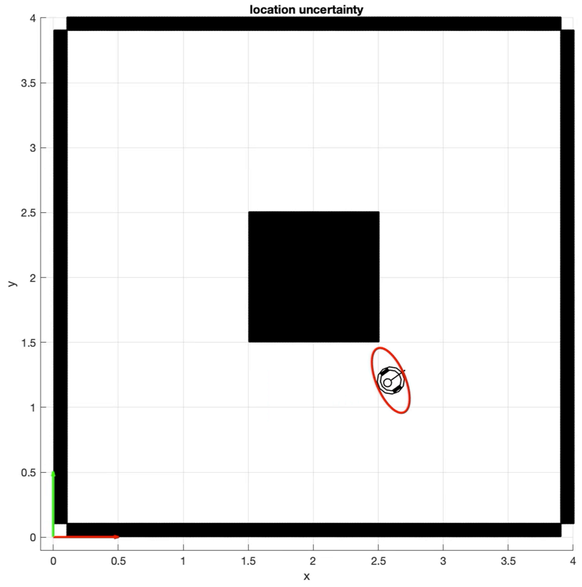
$$\Sigma_u = \begin{pmatrix} k_r |D_r| & 0 \\ 0 & k_l |D_l| \end{pmatrix}$$

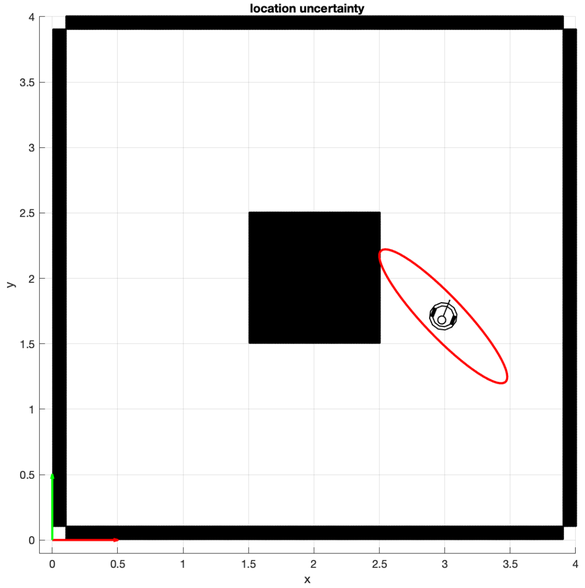
$$p_0 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$$

$$\Sigma_0 = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix}$$



initial estimate





spacial uncertainty

EKF - Correction / Update

observation model

$$o_i(k) = g_i\left(X(k), M\right) + h_i(k), \quad h_i(k) \sim N(0, R_i)$$

Correction / Update

noise covariance

Kalman gain

$$K = \underset{(3 \times n)}{\bar{\Sigma}_k} \underset{(3 \times 3)}{\nabla h^T} \Big(\underbrace{\underset{(n \times 3)}{\nabla h} \underset{(3 \times 3)}{\bar{\Sigma}_k} \underset{(3 \times n)}{\nabla h^T}}_{S_{(n \times n)}} + \underset{(n \times n)}{R_k} \Big)^{-1}$$

innovation covariance

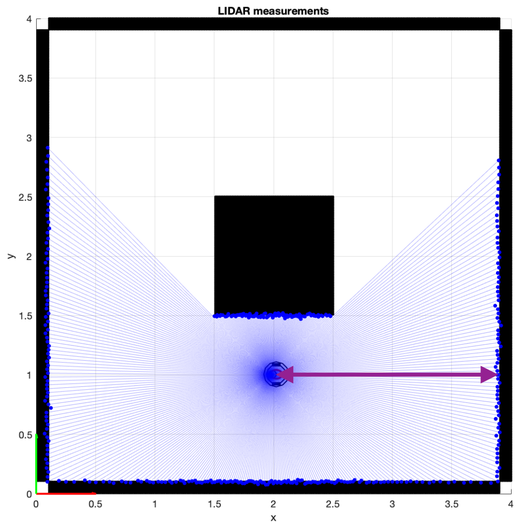
$$R_k = \begin{pmatrix} r_1 & 0 & 0 \\ 0 & r_i & 0 \\ 0 & 0 & r_n \end{pmatrix}$$

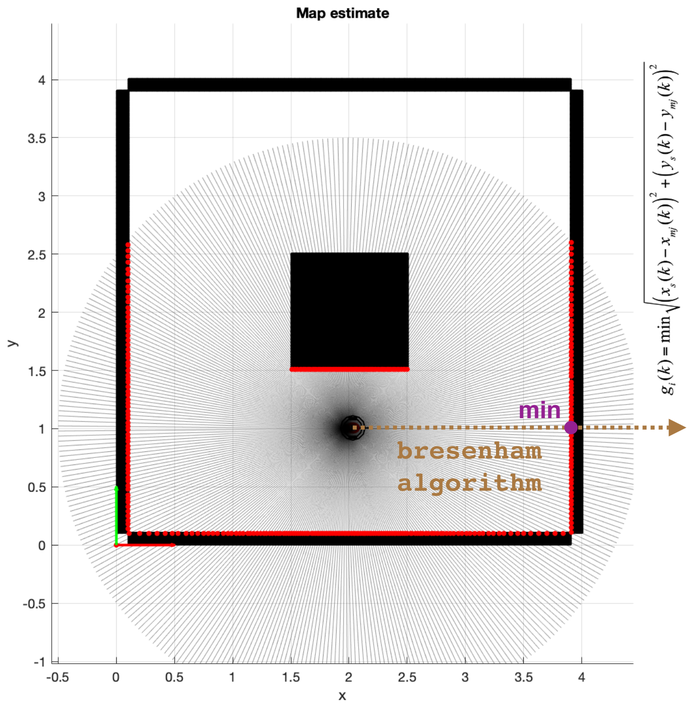
$$r_i = (3.5\ \%\ of\ z_k)^2$$

state vector

$$p_k = \underset{(3 \times 1)}{\bar{p}_k} + \underset{(3 \times n)}{K} \Big(\underbrace{\underset{(n \times 1)}{z_k} - \underset{(n \times 1)}{h(\bar{p}_k)}}_{v_{(n \times 1)}} \Big)$$

innovation





observation

predicted observation

z_k

$h(\bar{p}_k)$

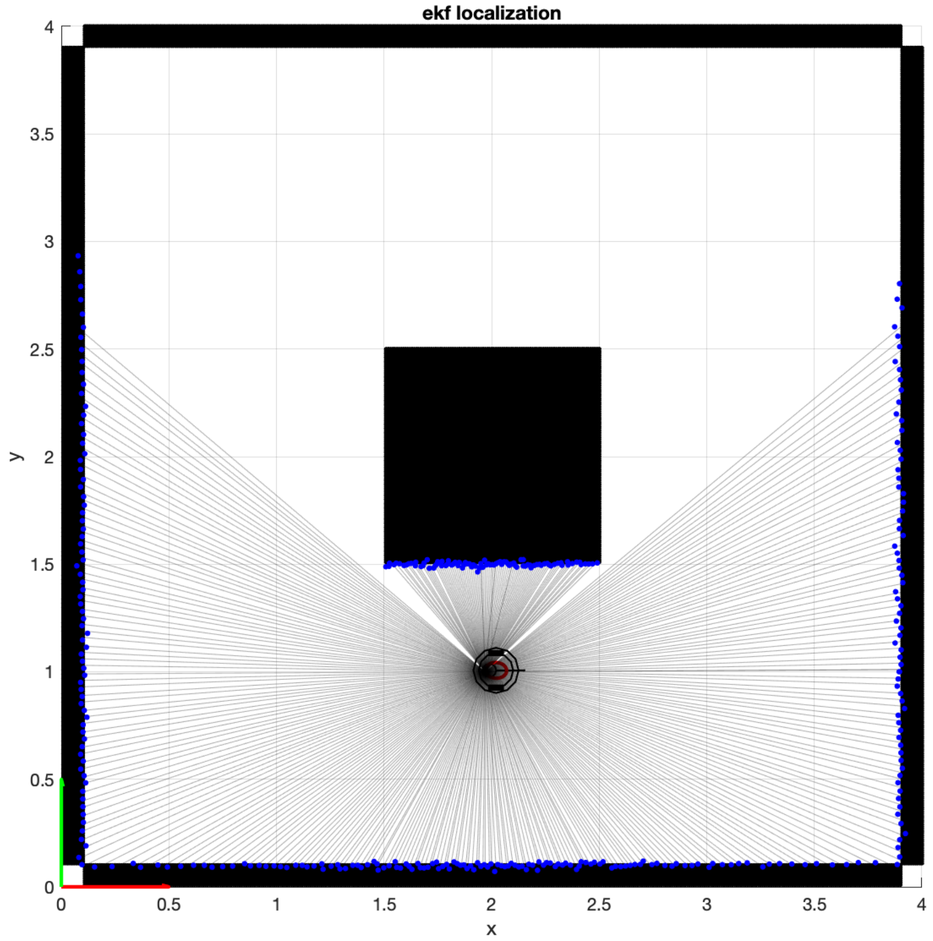
state covariance

$$\underset{(3 \times 3)}{\Sigma_k} = \Big(\underset{(3 \times 3)}{I_3} - \underset{(3 \times n)}{K} \underset{(n \times 3)}{\nabla h} \Big) \underset{(3 \times 3)}{\bar{\Sigma}_k}$$

Observations - Predictions Correspondences

Range Observations

z_k



Distance Prediction
(Map Query)

$h(\bar{p}_k)$

Outliers must to be removed from the innovation vector (v)

Observations - Predictions Correspondences (2)

Criteria for <u>accepting</u> matches (Mahalanobis distance):

$$v_i \; s_i^{-1} \; v_i < e^2$$

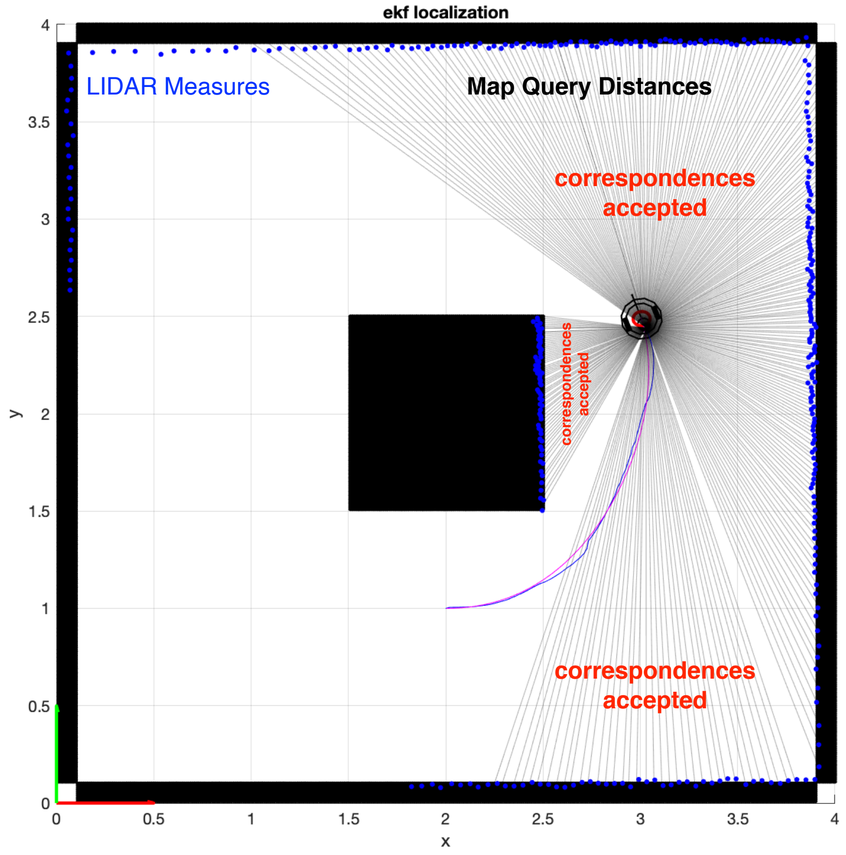
- Apply for each LIDAR measure individually (i)
- v_i innovation value (scalar).
- s_i innovation covariance (scalar query).

- Evaluate s_i, individually (per LIDAR measure):

$$s_i = \nabla h_i \; \bar{\Sigma}_k \; \nabla h_i^T + r_i$$

(1 x 1) (1 x 3) (3 x 3) (3 x 1) (1 x 1)

$$\nabla h_i = \begin{bmatrix} \frac{\partial h_i}{\partial x} & \frac{\partial h_i}{\partial y} & \frac{\partial h_i}{\partial \theta} \end{bmatrix}$$



Lab #4 StartUp Code

```
p = [2, 1, 0] ;          % state vector (default value for scmap)
tbot.setPose(.);         % set initial pose
Sigma = diag( [sigma_x  sigma_y deg2rad(sigma_theta)].^2 );        % state Covariance
tbot.setVelocity(.);     % assign velocities
tbot.initEncoders();     % startup the encoders module
r = rateControl(5);      % init rateControl obj - set loop at 5Hz
tic;
while (toc < 30)         % run for a given time (s)
        [.] = tbot.readEncoders();          % read data from encoders
        [.] = tbot.readLidar();             % read lidar data

        u = [delta_sr, delta_sl]';

        % Extended Kalman Filter Prediction
        [p, Sigma]= ekfPredict(p, Sigma, u, ... );

        % Measurement
        for i=1:1:360           % for each LIDAR reading
            % get observation
            observation = lddata.Ranges(i);

            % uncertainty is 3.5% of reading
            R = diag(.)

            % estimate distance according to map (predicted observation)
            estimate = g(p, ... )

            % innovation
            v_in = observation - estimate;

            % accept correspondences if:
            v_in * 1/S_i * v_in < e^2
        end

        % Extended Kalman Filter Update/Correction
        [p, Sigma, S] = ekfUpdate(p, Sigma, ... );

        % Display
        plot(.)
        waitfor(r);      % adaptive pause
end
tbot.stop();             % stop robot
```

Gradient Computation (w/ Symbolic MatLab Toolbox).

Demo example:

```
% clean memory
clear;

% initialize symbolic variables (type real)
syms x y theta f real;

% define generic expression function of the symbolic variables
f = [cos(theta), -sin(theta) x; sin(theta), cos(theta) y; 0, 0, 1] * [1; 2; 1];


% ---------------------------
% compute gradients
% ---------------------------

% derivative of (f) w.r.t. (x)
dfdx = diff(f,'x')

% derivative of (f) w.r.t. (y)
dfdy = diff(f,'y')

% derivative of (f) w.r.t. (theta)
dfdtheta = diff(f,'theta')


% ---------------------------
% print symbolic expressions
% ---------------------------
pretty(dfdx)
pretty(dfdy)
pretty(dfdtheta)
```

Console output:

```
>> pretty(dfdx)
/ 1 \
|   |
| 0 |
|   |
\ 0 /

>> pretty(dfdy)
/ 0 \
|   |
| 1 |
|   |
\ 0 /

>> pretty(dfdtheta)
/ - 2 cos(theta) - sin(theta) \
|                             |
|  cos(theta) - 2 sin(theta)  |
|                             |
\             0               /
```