

## Hamster Developer Manual

MODEL N° ACK-L360/6PI2LLC-REV4-O4.4





## Contents

Hamster User Manual .....	1
Introduction .....	3
Hamster Overview .....	3
Power switch and monitoring components .....	3
Hamster main components .....	5
Lidar .....	5
Camera (raspicam).....	5
CPU (ARM Linux).....	5
CPU (Arduino).....	6
Motors .....	6
Battery.....	6
Charger .....	6
ROS Architecture Overview .....	7
What is ROS? .....	7
Initial setup (accessing ROS Master) .....	7
Raspberry PI Master/Slave and ROS nodes layout .....	8
Hamster working environment .....	9
Installing and configuring ROS on the Linux Server .....	9
Installing and configuring the Linux Server OCU and Hamster software .....	9
Running the Hamster software on the Linux Server .....	9
Configuring the Hamster network .....	10
Configuring the Hamsters' IP .....	10
Example wandering code .....	11
Example blob detection code .....	14
Configuring Hamster Map .....	15
Configuring Hamster Location on the map .....	15
Hamster ROS topics.....	17
Hamster Simlation .....	20
Preparing The Simulation .....	16
Running The Simulation .....	17
Configuring More Hamsters on the Simulation .....	17
Hamster LLC rev3.....	20

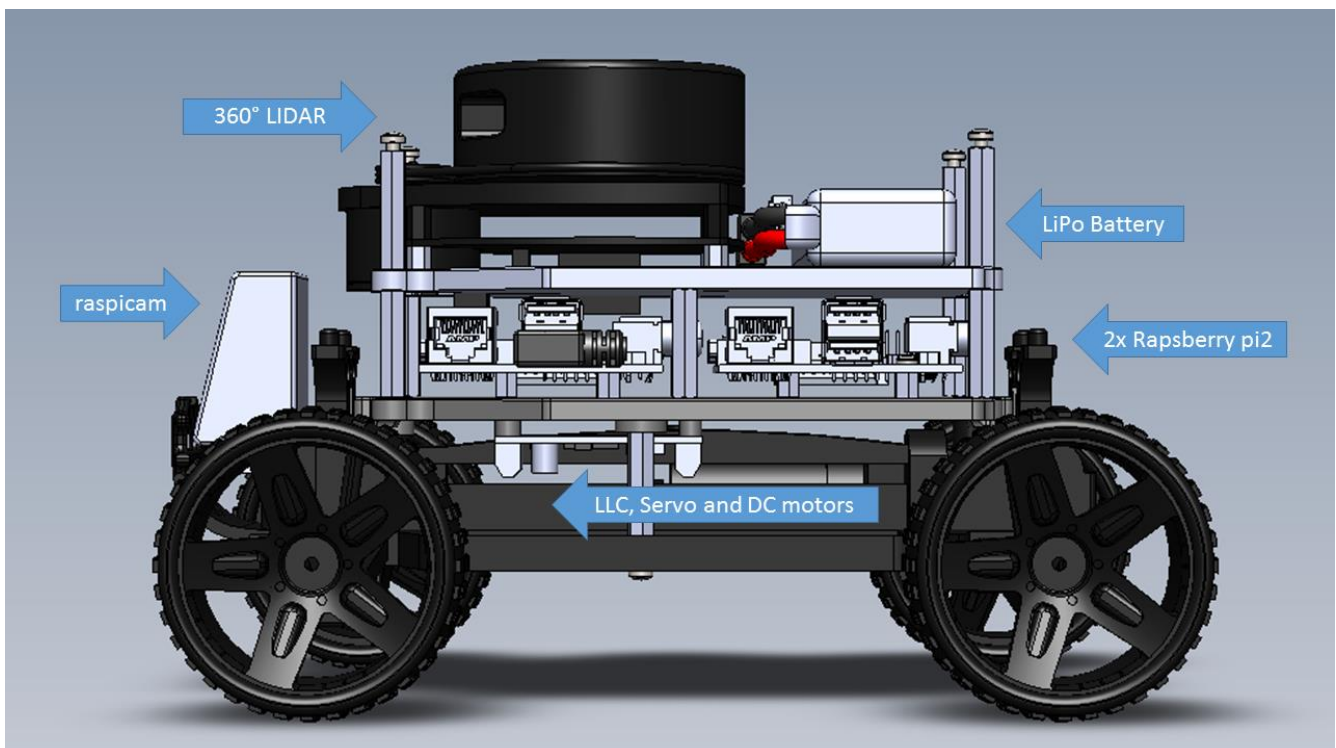
## Introduction

Hamster robot is a robotic research platform focused on autonomy.

## Hamster Overview

The hamster platform arrives with various components pre-installed:

- A 6m range 360 Lidar supported by the ROS community here <http://wiki.ros.org/rplidar>.
- An HD capable camera supported by ROS [https://github.com/fpasteau/raspicam\\_node](https://github.com/fpasteau/raspicam_node)
- 2 Raspberry PI2 boards with Linux (Debian/Raspbian) and ROS indigo
- A Low Level controller with dedicated hardware for power distribution and monitoring



## Charging, Power switch and monitoring components

The Hamster robot is shipped with powerful LIPO batteries which are fast to charge and provide plenty of power needed for the platform operation over long periods and with demanding sensors.

The use of LIPO batteries however requires special attention in order to prevent the discharge of the batteries (since once LIPO batteries are fully discharged they cannot be re-charged again).

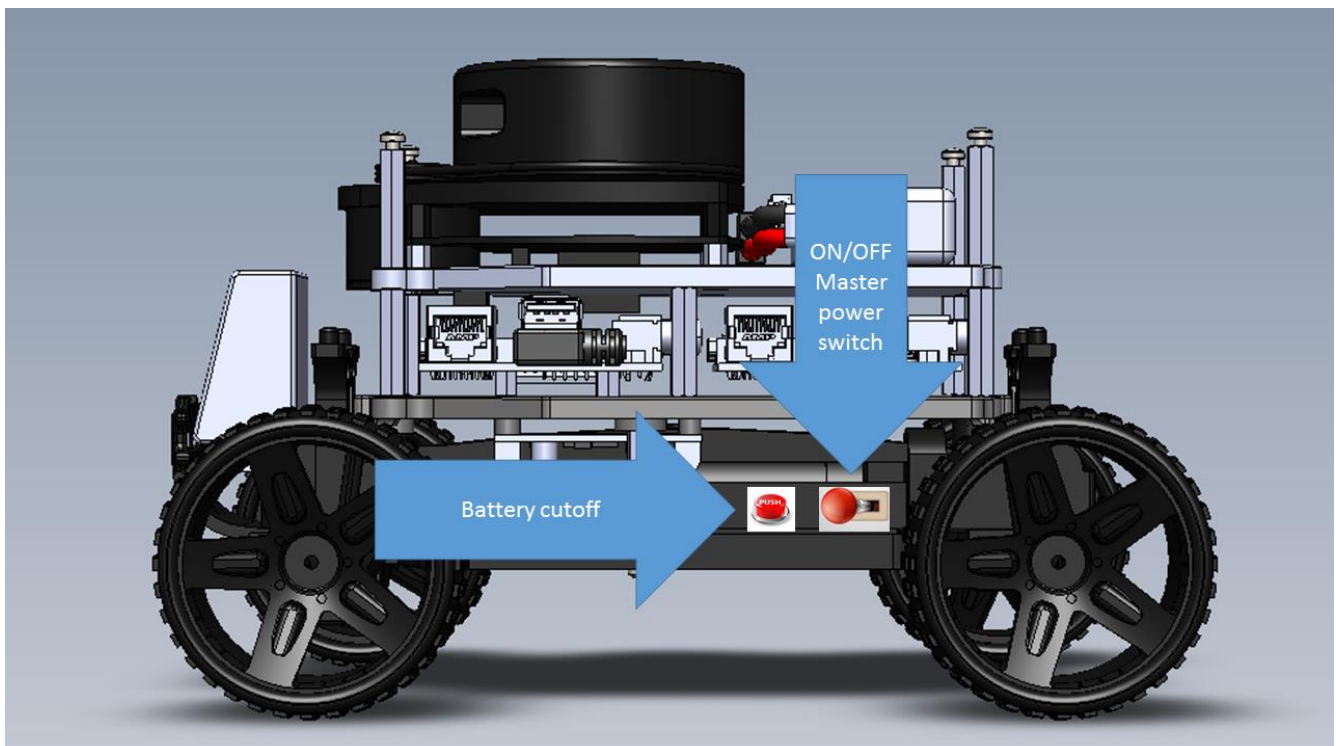
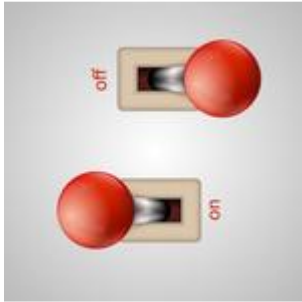
Hence, the Hamster robot arrives with a double switch mechanism.

They are referred as the *master power switch* and the *battery cutoff button*.



The master power switch is in charge of the main power relay. **The Hamster can be charged only when the master power switch is in it's OFF state.** Charging is done through the JST connector at the rear part of the robot. No need to take out the batteries or open the robot.

Once the switch is ON, the battery monitoring mechanism can be started by pressing the battery cutoff button next to it.



## Hamster main components

### *Lidar*

- Range 6m 360deg
- Angular resolution 1°
- Scan Frequency 5Hz
- Power 5VDC
- USB connection to computing module



Figure 1 Lidar

### *Camera (raspicam)*

- five megapixel fixed-focus camera
- Supports 1080p30, 720p60 and VGA90 video modes
- Supports stills capture.
- It attaches via a 15cm ribbon cable to CSI port on
- Can be accessed through the MMAL and V4L APIs, and there are numerous third-party libraries built for it
- Including a Python library

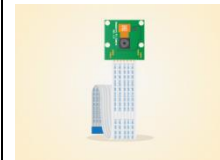


Figure 2 Camera module

### *CPU (ARM Linux)*

- 2 x ARM CPU
- 2 x 1Ghz quad core
- 2 x 1G RAM
- 2 x Videocore 4 GPU
  - .1. The GPU is capable of BluRay quality playback
  - .2. Using H.264 at 40MBits/s.
  - .3. Fast 3D core
  - .4. Supplied OpenGL ES2.0 and OpenVG libraries
- 32Gb SD
- More than 52 dedicated GPIO pins
  - .1. UART
  - .2. i2c bus
  - .3. SPI bus with two chip selects
  - .4. i2s audio,
  - .5. 3v3, 5v, and ground.
  - .6. DIO
  - .7. PWM
  - .8. ADC
- 8 USB
- Current stabilization to 5v for up to 7A
- Wifi




--	--

### **CPU (Arduino)**

<ul style="list-style-type: none"> <li>• Arduino pro mini 5v</li> <li>• Low Level control</li> <li>• Battery state monitor</li> </ul>	
---	---

### **Motors**

<ul style="list-style-type: none"> <li>• 4.4:1 Metal Gearmotor 25Dx48L mm HP</li> <li>• EMAX Servo motor 2kg Turque ES08MA II With Metal Gear</li> </ul>	
--	--

### **Battery**

<ul style="list-style-type: none"> <li>• Lipo</li> <li>• 2200MAh</li> <li>• 1H continuous operation</li> <li>• 3H at standby</li> <li>• Extendable</li> </ul>	
---	---

### **Charger**

<ul style="list-style-type: none"> <li>• Provided with the System</li> </ul>	
--	--

## ROS Architecture Overview

Hamster robot arrives with Linux (ARM) and ROS pre-installed.  
Thus, the best method of communication with the robot is through ROS.

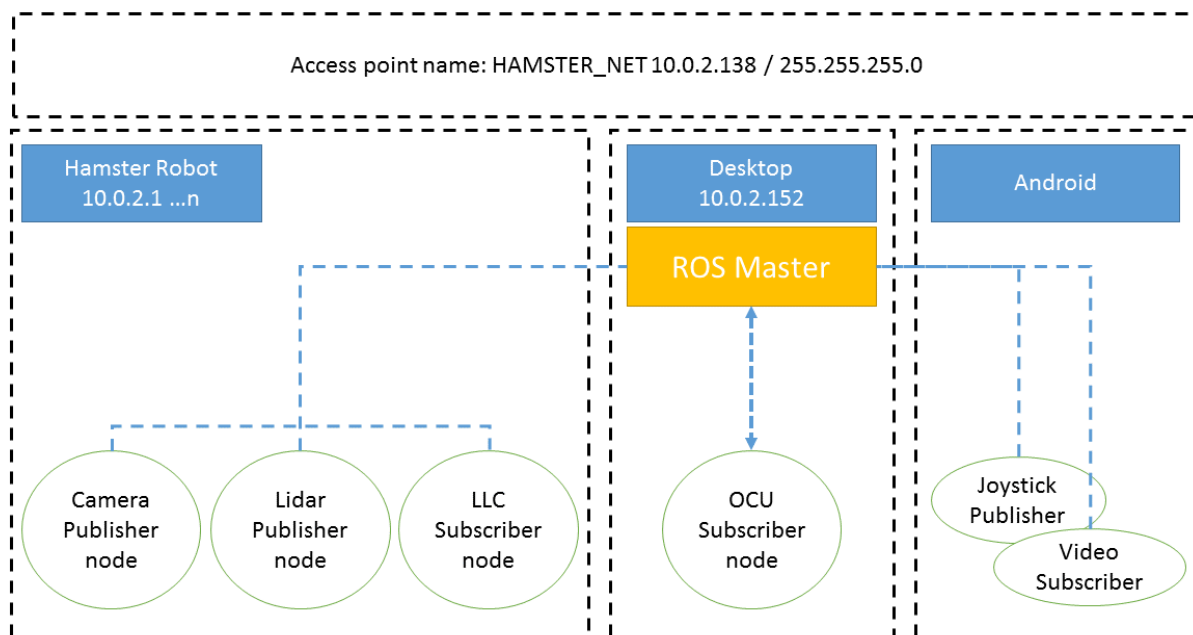
### ***What is ROS?***

ROS (Robot Operating System) is a BSD-licensed system for robotic components development.

A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model.

### ***Initial setup (accessing ROS Master)***

The ROS master is very similar to a DNS server in ROS. It keeps the IP/Port of the various nodes (components) in the system and enables the initial connection setup between them.

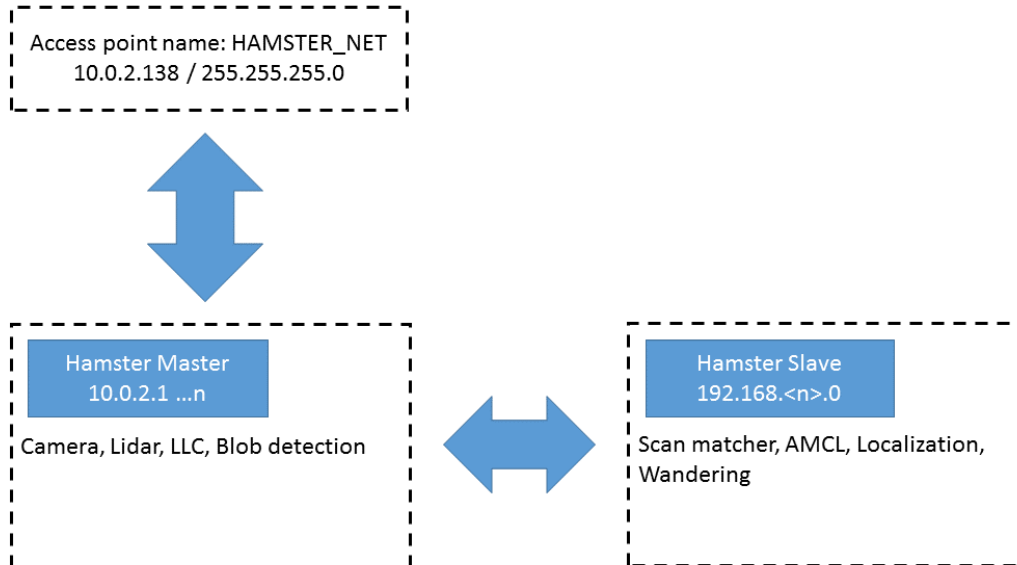


Once the connection is made between the publisher and subscriber, the communication turns peer to peer and is direct over TCP (sensor data does not pass through the ROS master).

## Raspberry PI Master/Slave and ROS nodes layout

Hamster uses a dual system configuration. The two CPU's are connected over Ethernet and form a local network. One of the server has two network interfaces, one connected to the local network and one serves as a gateway for the outside communication.

The software installed on the two servers is identical, but a different ROS launch is being executed. This can be configured in `~/ros` script (which is being run at startup by a cron service)



As noted above, by default the system loads different processes on the master and slave servers. This can be configured in the `hamster_launch` ROS package.

### Hamster Master

- Camera, Lidar, LLC, Blob detection
- In `~/ros` the following launch is configured  
`roslaunch hamster_launch platform.launch`

### Hamster Slave

- Scan matcher, AMCL, Localization, Wandering
- In `~/ros` the following launch is configured  
`roslaunch hamster_launch localization.launch`





## Hamster working environment

### *Installing and configuring ROS on the Linux Server*

ROS uses a master server program in order to connect between different ROS node programs.

In order to install ROS on a Linux (Ubuntu) machine follow these links

- <http://wiki.ros.org/indigo/Installation/Ubuntu>

Once ROS is installed, you will need to configure your system, follow the tutorials.

- <http://wiki.ros.org/roscore>
- <http://wiki.ros.org/ROS/Tutorials>

### *Installing and configuring the Linux Server OCU and Hamster software*

From the terminal run the following to update required libraries

```
sudo apt-get install ros-indigo-map-server
```

Download the code and compile it

```
cd ~/
mkdir hamster_ws/src -p
cd hamster_ws/src
git clone https://github.com/cogniteam/hamster\_server.git
cd ..
catkin_make
source devel/setup.bash
```

Add the following line in ~/.bashrc

```
source ~/hamster_ws/devel/setup.bash
```

### *Running the Hamster software on the Linux Server*

Configure the server's IP to be 10.0.2.152 .

Set the environmental parameters – ROS\_MASTER\_URI and ROS\_IP, using the following commands:

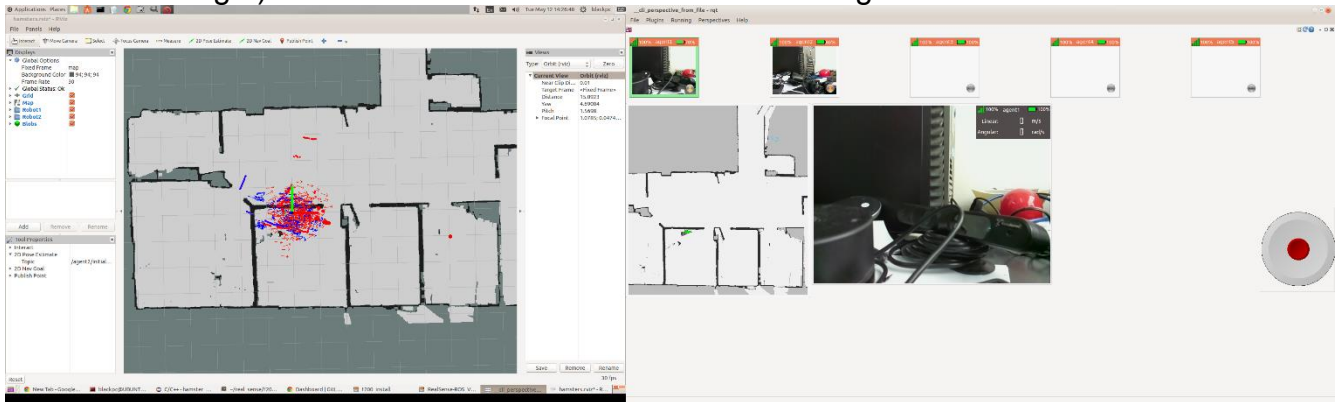
```
export ROS_MASTER_URI=http://10.0.2.152:11311
export ROS_IP=10.0.2.152
```

To launch the server's node, run the following command:

```
roslaunch hamster_server_launch server.launch
```



This command will start rviz (as seen below on the left) and Cogniteam's OCU (as seen below on the right). The two hamsters should start streaming information to the OCU.



## ***Configuring the Hamster network***

Each Hamster connects to the Linux Server through a router.  
Wireless configuration on the router:

Name: hamster\_net  
Security: WPA2 personal  
Password: hamsterHAMSTER

Netmask: 255.255.255.0  
Router's IP: 10.0.2.138

Also, in order to reach the slave raspberry pi on each Hamster, configure the routing table as following:

Go to Advanced Routing → Static Routing List → Add...

For each robot (numbered <n>), add the corresponding entry:

Destination IP: 192.168.<n>.0  
Subnet Mask: 255.255.255.0  
Default gateway: 10.0.2.<n>

## ***Configuring the Hamsters' IP***

**Hamster robots arrive with a pre-configured specific IP's. Unless you know what you are doing, this should not be changed.**

Robot 1 configuration:  
Master raspberry pi IP: 10.0.2.1 or 192.168.1.1  
Slave raspberry pi IP: 192.168.1.2

Robot 2 configuration:



Master raspberry pi IP: 10.0.2.2 or 192.168.2.1

Slave raspberry pi IP: 192.168.2.2

The Hamster's connection configurations can be changed in the `/etc/network/interfaces`, and `/wpa_supplicant/wpa_supplicant.conf`.

SSH connection details:

Username: pi

Password: hamster

On the home directory of each robot (`/home/pi/`), a file named `hamster.config` contains details for Hamster's IP and ID and camera configurations such as width, height, fps and quality.

## ***Hamster mode switch***

Hamster arrives with 3 pre-set modes:

- Slam – builds an indoor map
- Localization – Localizes indoor on a known map
- Outdoor – Uses GPS and IMU / Odometry to better localize on a global coordinate system

If you want to manually switch between modes (localization/mapping/outdoor), you can send a string message to **/mapping\_command** topic containing "slam" or "localization" string.

For example:

```
rostopic pub /mapping_command std_msgs/String "data: 'slam'"
```

or

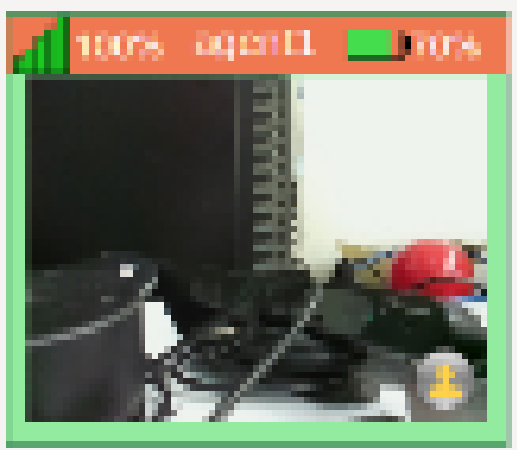
```
rostopic pub /mapping_command std_msgs/String "data: 'localization'"
```

or

```
rostopic pub /mapping_command std_msgs/String "data: 'outdoor'"
```

## Example wandering code

The wandering example can be started by pressing on the small icon (on the bottom right corner) in the camera panel (in the OCU).



For wandering Hamster uses the ROS Navigation stack (documented <http://wiki.ros.org/navigation>). Steering is done using trajectories and an Ackerman steering model.

```
void Wandering::spin() {
    ros::NodeHandle nodePrivate("~");

    srand(time(0));

    LaserScanDataSource* laserScanDataSource = new LaserScanDataSource(nodePrivate, "/scan");
    CostMap costMap(laserScanDataSource, new RosParametersProvider());

    /**
     * Publishers
     */
    ros::Publisher mapPublisher = nodePrivate.advertise<nav_msgs::OccupancyGrid>("costmap", 1, false);
    ros::Publisher pathPublisher = nodePrivate.advertise<nav_msgs::Path>("path", 1, false);
    ros::Publisher bestPathPublisher = nodePrivate.advertise<nav_msgs::Path>("path_best", 1, false);
    ros::Publisher ackermannPublisher =
nodePrivate.advertise<ackermann_msgs::AckermannDriveStamped>("/ackermann_cmd", 1, false);
    ros::Subscriber stateSubscriber = nodePrivate.subscribe(string("/decision_making/" + robotId_ +
"/events"), 1, &Wandering::stateCallback, this);

    createTrajectories(3.0, 0.1);

    ros::Rate rate(7);

    while (ros::ok()) {
        ros::spinOnce();

        if (!enabled_) {

            if (publishStop_) {
```



```
ackermannPublisher.publish(ackermann_msgs::AckermannDriveStamped());
publishStop_ = 0;
```

```
}
```

```
rate.sleep();
continue;
```

```
}
```

```
/**
```

```
 * Evaluate trajectories
```

```
 */
```

```
TrajectoryMatch::Ptr bestMatch = chooseBestTrajectory(costMap);
```

```
/**
```

```
 * Publish all paths
```

```
 */
```

```
for (int i = 0; i < frontTrajectories_>size(); ++i) {
    pathPublisher.publish(*frontTrajectories_[i]->getPath(true, baseFrameId_));
    boost::this_thread::sleep(boost::posix_time::milliseconds(1));
}
```

```
/**
```

```
 * Publish best matched trajectory
```

```
 */
```

```
bestPathPublisher.publish(bestMatch->getTrajectory()->getPath(true, baseFrameId_));
```

```
/**
```

```
 * Publish velocity command
```

```
 */
```

```
ackermannPublisher.publish(
    bestMatch->getTrajectory()
    ->getMotionModelAs<AckermannModel>()
    ->getAckermannMessage());
```

```
/**
```

```
 * Publish local cost map
```

```
 */
```

```
mapPublisher.publish(costMap.getOccupancyGrid());
```

```
rate.sleep();
```

```
}
```

```
}
```

Trajectories are evaluated and kept in a sorted Set.

```
void Wandering::createTrajectories(double simulationTime, double granularity) {
    TrajectorySimulator trajectorySimulator(simulationTime, granularity);
```

```
    Trajectory::Ptr trajectory;
```

```
    static const double MIN_ANGLE = -0.30;
```

```
    static const double MAX_ANGLE = 0.30;
```

```
    static const int    TRAJECTORIES = 5;
```

```
    static const double ANGLE_RANGE = -MIN_ANGLE + MAX_ANGLE;
```

```
    static const double ANGLE_STEP = ANGLE_RANGE / (TRAJECTORIES - 1);
```

```
    frontTrajectories_ = Trajectory::VectorPtr(new Trajectory::Vector());
```



```
for (double i = 0; i < TRAJECTORIES; ++i) {

    const double currentAngle = MIN_ANGLE + ANGLE_STEP * i;

    trajectory = trajectorySimulator.simulate(new AckermannModel(0.165, 0.48, currentAngle));
    trajectory->setWeight(1.0 - (fabs(currentAngle) / (ANGLE_RANGE)));
    frontTrajectories_->push_back(trajectory);
}

frontTrajectory_ = (*frontTrajectories_)[TRAJECTORIES / 2];
}
```

## Example blob detection code

For blob detection Hamster uses in this example the cmvision library integrated in ROS and documented here <http://wiki.ros.org/cmvision>

Our code main logic can be found in hamster\_blob\_detection\_node.cpp

```
void imageCallback(const sensor_msgs::ImageConstPtr& image) {
    if ((ros::Time::now() - last_blob_detection_).toSec() < detection_delay_)
        return;

    last_blob_detection_ = ros::Time::now();
    cmvision::Blobs blobs = cmvisionDetector.imageCB(image);
    ROS_INFO("Blobs detected = %i", blobs.blobs.size());
    if (blobs.blobs.size() >= 1) {
        const cmvision::Blob& blob = blobs.blobs[0];
        const double blobWidth = blob.right - blob.left;
        const double horizontalFov = 0.785398163;
        const double blobCenter = (blob.right + blob.left) / 2.0;

        double distance =
            ( (0.13 * image->width) / (2.0 * blobWidth) ) * tan(M_PI_2 - (horizontalFov / 2.0));
        double bearing =
            atan((2 * (image->width / 2 - blobCenter) * tan(horizontalFov / 2)) / image->width);

        tf::Transform blobRotation = tf::Transform::getIdentity();
        blobRotation.setRotation(tf::createQuaternionFromYaw(bearing));

        tf::Transform blobOffsetTf;
        blobOffsetTf.setOrigin(tf::Vector3(distance, 0, 0));
        blobOffsetTf.setRotation(tf::Quaternion::getIdentity());

        tf::Vector3 blobPosition = blobRotation * blobOffsetTf * tf::Vector3(0, 0, 0);

        visualization_msgs::Marker marker;
        marker.action = visualization_msgs::Marker::ADD;
        marker.color.a = 1;
        marker.color.r = 1;
        marker.header.frame_id = frame_id_;
        marker.header.stamp = ros::Time::now();
    }
}
```



```
marker.lifetime = ros::Duration();
marker.id = rand();
marker.ns = "/";
marker.pose.position.x = blobPosition.x();
marker.pose.position.y = blobPosition.y();
marker.pose.orientation.w = 1;
marker.scale.x = 0.14;
marker.scale.y = 0.14;
marker.scale.z = 0.14;
marker.type = visualization_msgs::Marker::SPHERE;

markerPublisher.publish(marker);
}
}
```

Basically, we publish over ROS markers found by the CMU library. Those markers can be viewed in rviz.

## ***Configuring Hamster Map***

The Hamster robot arrives with a localization node based on amcl. This node uses a bitmap map to localize. To set the working map, go to the maps folder in the hamster\_server\_launch package, using:

```
roscd hamster_server_launch/maps
```

Copy the map image to that folder.

Also, edit the launch file to work with the new map: cd to the hamster\_server\_launch/launch directory, and edit server.launch . Change the field of map\_server, enter the wanted yaml file path. (e.g.: \$(find hamster\_server\_launch/maps/cogniteam.yaml) )

## ***Configuring Hamster Location on the map***

To set position for an agent, use Rviz's 2D Pose Estimation, which is located on the top bar.

Afterwards, click on the map in the desired position, drag to set heading and release when needed.

To set position for agent N, change to the appropriate topic using:  
Tool properties → Pose estimation → Topic : /agentN/initialpose

## ***Hamster map reset***

In order to reset the map, you should send a string message "reset" to /agent1/syscommand topic: **rostopic pub /agent1/syscommand std\_msgs/String "data: 'reset'"**





## ***Hamster ROS topics***

Topic	Description
/agentN/scan	The laser scan from the LIDAR.
/agentN/image/compressed	Camera video stream
/agentN/battery	Battery voltage (ranges from 3.2v to 4.2v)
/agentN/rssi	Signal strength of WIFI (ranges from 0-100%)
/agentN/ackermann_cmd	Velocity command (only speed and steering_angle are used)
/agentN/amcl_pose	Absolute position published by AMCL node (localization)
/agentN/pose2D	Position estimated by laser scan matcher node
/agentN/imu	Compass, Accelerometer
/agentN/fix	GPS lat/lon WGS84, covariance, altitude
/decision_making/agentN/events	Wandering control topic (RESUME for start, PAUSE for stop)
/tf	All transformation (including robot's position)
/blobs	Blobs detected
/map	A map used for localization



## ***Hamster Simulation***

### ***Preparing the simulation***

It is assumed that ROS is installed on indigo with the full desktop version, Please refer back to installing and configuring ROS before moving on.

Configuring Hamster for gazebo requires the following libraries to be installed

```
sudo apt-get install ros-indigo-controller-manager ros-indigo-controller-interface ros-indigo-gazebo-ros ros-indigo-gazebo-ros-control ros-indigo-gazebo-ros-pkgs ros-indigo-ackermann-msgs ros-indigo-map-server ros-indigo-ros-control ros-indigo-ros-controllers
```

After downloading those packages, make sure that gazebo is installed and running, it should've been installed along with ros-indigo-desktop-full package

```
gazebo
```

Once everything is set up, clone and create the gazebo workspace

```
cd ~/
mkdir hamster_gazebo_ws/src -p
cd hamster_gazebo_ws/src
git clone https://github.com/cogniteam/hamster\_gazebo.git
cd hamster_gazebo
mv * ../
cd ..
rm -rf hamster_gazebo
cd ..
catkin_make -j1
```

This should compile everything needed to run the simulation on gazebo

## ***Running the simulation***

First, make the created workspace your source by doing  
`source ~/hamster_gazebo_ws/devel/setup.sh`

Before actually running the simulation, make sure that ros is configured to work locally, meaning both ROS\_IP and ROS\_MASTER\_URI are configured locally you can configure that by doing the following:

```
export ROS_IP=your_local_ip
export ROS_MASTER_URI=http://your_local_ip:11311
```

The local ip can be obtained via the ifconfig command.  
To make sure that it is configured properly, run the following

```
echo $ROS_IP
should print your IP
echo $ROS_MASTER_URI
should print the uri with your IP
```

Afterwards, to run the gazebo simulation, run the following

```
roslaunch hamster_sim hamster.sh
```

This will open up gazebo with the simulation and the OCU for controlling the hamsters  
It should look like the following



## ***Configuring more hamsters on the simulation***

Right now the simulation is configured for two hamsters, In order to add more agents to the simulation



The following command can be changed into:

```
roslaunch hamster_sim hamster.sh -r=3 -rp=0,1:1,0:0,0
```

Where  $r$  is the amount of robots (Up to 5) and  $rp$  is the starting position for each robot on the gazebo

## Hamster LLC rev4

Hamster Low Level Control (LLC) enables to program the low level aspects of the Hamster Robots such as interfacing the motors drivers the power management and the motors (DC and Servo) directly, without using the ROS interface. The firmware installed on the Hamster is open sourced, corresponds LLC rev4 and is included as part of the Hamster SDK.

For any practical use there should be no need to program the Hamster using the Arduino interface, **doing so does brake the warranty and might brick the Robot.**

For flushing the LLC firmware you will need an FTDI board (available here [https://www.sparkfun.com/products/9716?\\_ga=1.249990533.2042159565.1430239634](https://www.sparkfun.com/products/9716?_ga=1.249990533.2042159565.1430239634) Not included)



Connect the pins directly to the Arduino pro mini