Project ‘Mr. Mess(enger)’

By Alejandro Alonso
https://automacomp.blogspot.com
April 2018

Purpose

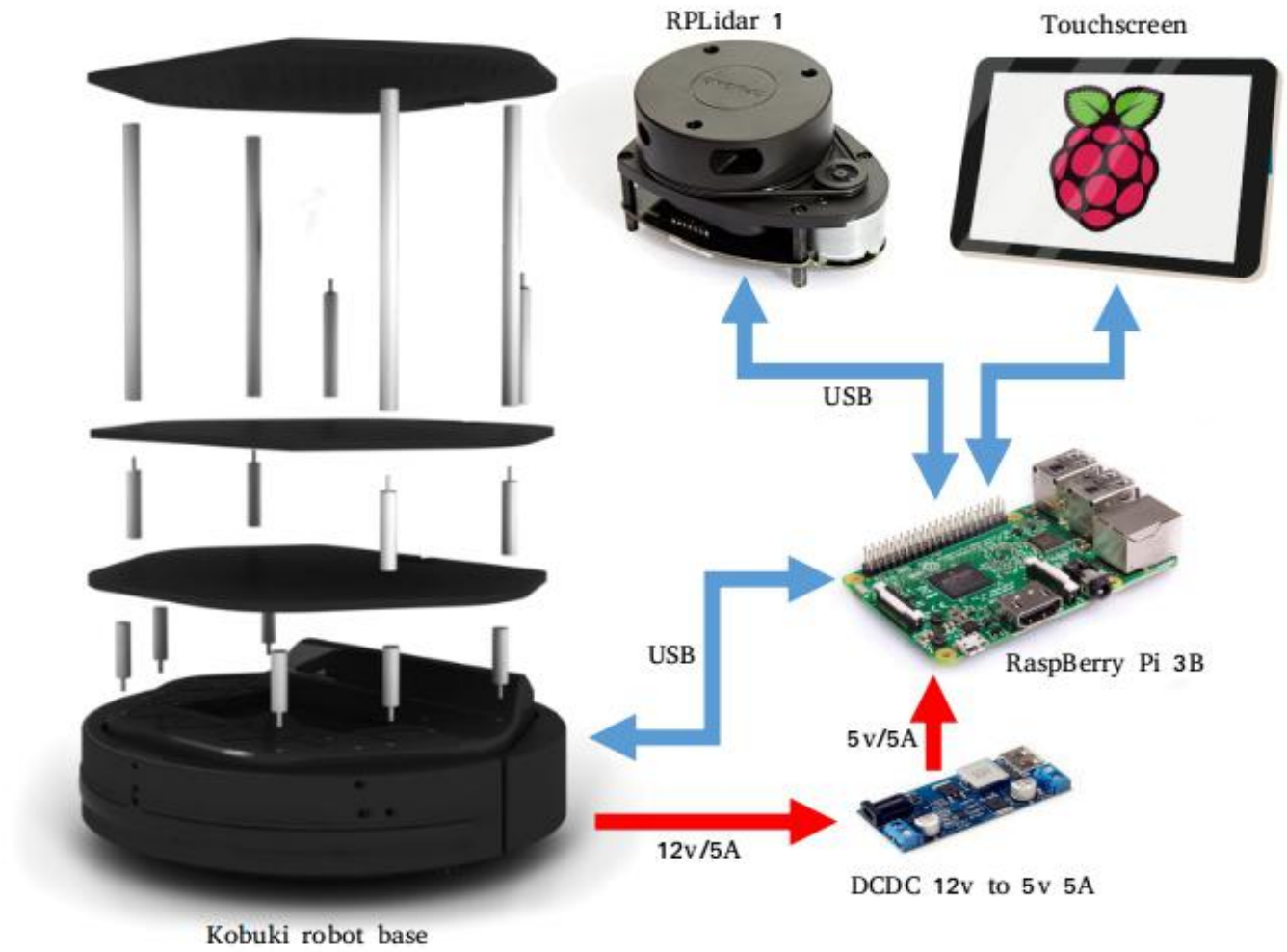
The purpose of this project is to configure and program a robot with ROS to go autonomously to specific locations on demand. A touch screen will show the locations. The user could put some weight on the robot and push the touchscreen for another destination to deliver the goods. This might be useful for home or office.

Index

The hardware

We are using a Turtlebot 2 robot with Kobuki base[i] as robot platform, a RPLidar 1 as Lidar, a Raspberry Pi 3B as embedded computer, a 7” touchscreen and a DCDC regulator 12v to 5v for powering the Raspberry Pi and Lidar from the robot batteries.

The cost of these items is under 1,800 us$



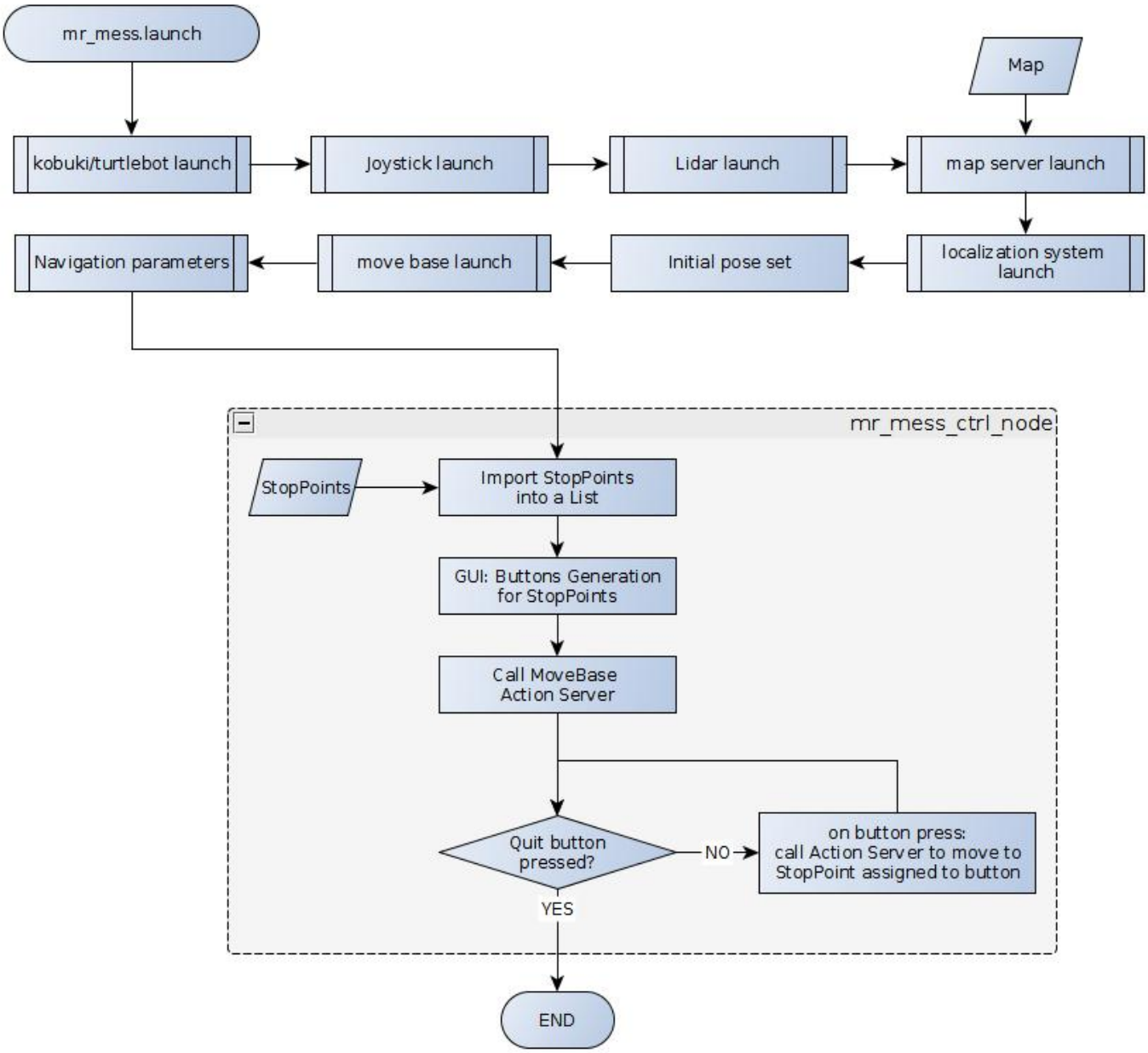
List of main components:

Component	Reference	Price	Link
Robot Platform	Turtlebot 2	1,100 us$	https://www.clearpathrobotics.com/turtlebot-2-open-source-robot/
Onboard computer	Raspberry Pi 3B	35 us$	https://www.adafruit.com/product/3055
SD card	16GB Ultra Micro SDHC UHS-I/Class 10 Card	9 us$	https://www.amazon.com/SanDisk-Ultra-Micro-Adapter-SDSQUNC-016G-GN6MA/dp/B010Q57SEE
Touch Screen	Daughter Board, Raspberry Pi 7" Touch Screen Display, 10 Finger Capacitive Touch	90 us$	https://www.raspberrypi.org/products/raspberry-pi-touch-display/
Lidar	RPLidar A1 (no longer available. A3 version is compatible)	450 us$	http://www.slamtec.com/en/lidar/a1
DCDC Regulator	D24V50F	15 us$	https://www.pololu.com/product/2851

The Software

The system uses ROS Kinetic over Ubuntu Mate Linux distribution on a Raspberry Pi 3B. See Annex 1 at the end of this document for instructions on how to install Ubuntu Mate, ROS Kinetic, Turtlebot/Kobuki ROS drivers and RPLidar ROS drivers.

Here is the Flowchart of the application developed. All is under a package called **mr_mess** and is executed from the launch file **mr_mess.launch** by using the command **roslaunch mr_mess mr_mess.launch**



Let’s see the different parts:

kobuki launch

turtlebot_bringup provides roslaunch scripts for starting the TurtleBot base (kobuki) functionality.

Inside the mr_mess.launch file it is called as:

<include file="$(find turtlebot_bringup)/launch/minimal.launch"/>

More information on this script: http://wiki.ros.org/turtlebot_bringup

Joystick launch

We configure our system to use a logitech remote Joystick for manual control of the robot. The manual control will always have priority over automated control.

Inside the mr_mess.launch file it is called as:

<include file="$(find turtlebot_teleop)/launch/logitech.launch"/>

Lidar launch

This script launch the Lidar. In out case RPLidar v1.

Inside the mr_mess.launch file it is called as:

<include file="$(find rplidar_ros)/launch/rplidar_setup2.launch"/>

Usually this launch file is called rplidar.launch, but as I use more than one robot with compatible RPLidars, I created one different Launch file per Lidar, with different frame.

For more information on this script, go to the Annex1, RPLidar setup.

map server launch

map_server provides the map_server ROS Node, which offers map data as a ROS Service. This map will be used by the navigation stack.

Inside the mr_mess.launch file it is called as:

<node name="map_server" pkg="map_server" type="map_server" args="$(find mr_mess)/maps/map.yaml" />

map.yaml refers to the map used for the navigation. This map is created at the beginning of the system setup. See section ‘How to create the map’ further ahead.

More information on this script: http://wiki.ros.org/map_server

localization system launch & Initial Pose

AMCL uses a particle filter to track the position of the robot.
The AMCL (Adaptive Monte Carlo Localization) package provides the **amcl** node, which uses the MCL system in order to track the localization of a robot moving in a 2D space. This node subscribes to the data of the laser, the laser-based map, and the transformations of the robot, and publishes its estimated position in the map. On startup, the amcl node initializes its particle filter according to the parameters provided.

Inside the mr_mess.launch file it is called as:

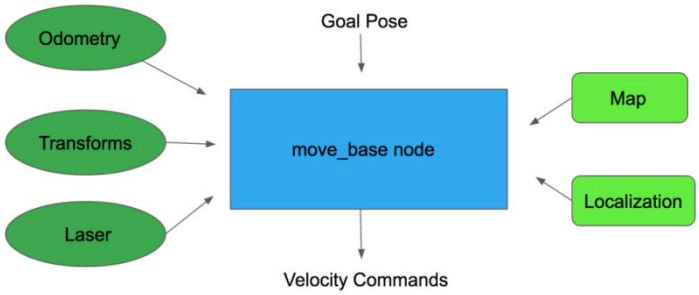
```
<!-- Launch AMCL localization system & Initial Pose Set-->
<arg name="custom_amcl_launch_file" default="$(find
    turtlebot_navigation)/launch/includes/amcl/amcl.launch.xml"/>
<arg name="initial_pose_x" default="6.1"/>
<arg name="initial_pose_y" default="9.7"/>
<arg name="initial_pose_a" default="3.14"/>
<include file="$(arg custom_amcl_launch_file)">
    <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
    <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
    <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
</include>
```

Previous code launch the localization system as well as define the pose (location and orientation) of the robot in the map, in the moment of running the system. Usually this is the location of the charging station. The location is defined in initial_pose_x and initial_pose_y and the orientation (in radians) in initial_pose_a. For more info on this, go to the section ‘Identify key Stop_Points’

For more information on amcl, go to http://wiki.ros.org/amcl

move base launch

The move_base package provides an implementation of an action (see the actionlib package) that, given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task.



Inside the mr_mess.launch file it is called as:

```
<!-- Launch Move base server-->
<arg name="custom_param_file" default="$(find
    turtlebot_navigation)/param/r200_costmap_params.yaml"/>
<include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml">
    <arg name="custom_param_file" value="$(arg custom_param_file)"/>
</include>
```

More information: http://wiki.ros.org/move_base

Navigation parameters

This section set some navigation parameters defined in the *mr_mess_nav_params.yaml* file

Inside the mr_mess.launch file it is called as:

```
<!-- Parameters -->
<rosparam command="load" file="$(find mr_mess)/params/mr_mess_nav_params.yaml" />
```

These parameters are for the navigation stack as well as for the manual control with joystick, limiting velocities and accelerations:

```
#navigation parameters
/navigation_velocity_smoother/speed_lim_v: 0.2
/navigation_velocity_smoother/speed_lim_w: 0.4
/navigation_velocity_smoother/accel_lim_v: 0.3
/navigation_velocity_smoother/accel_lim_w: 0.4
/navigation_velocity_smoother/decel_factor: 0.5
/teleop_velocity_smoother/speed_lim_v: 0.2
/teleop_velocity_smoother/speed_lim_w: 0.4
/teleop_velocity_smoother/accel_lim_v: 0.3
/teleop_velocity_smoother/accel_lim_w: 0.4
/teleop_velocity_smoother/decel_factor: 0.5
```

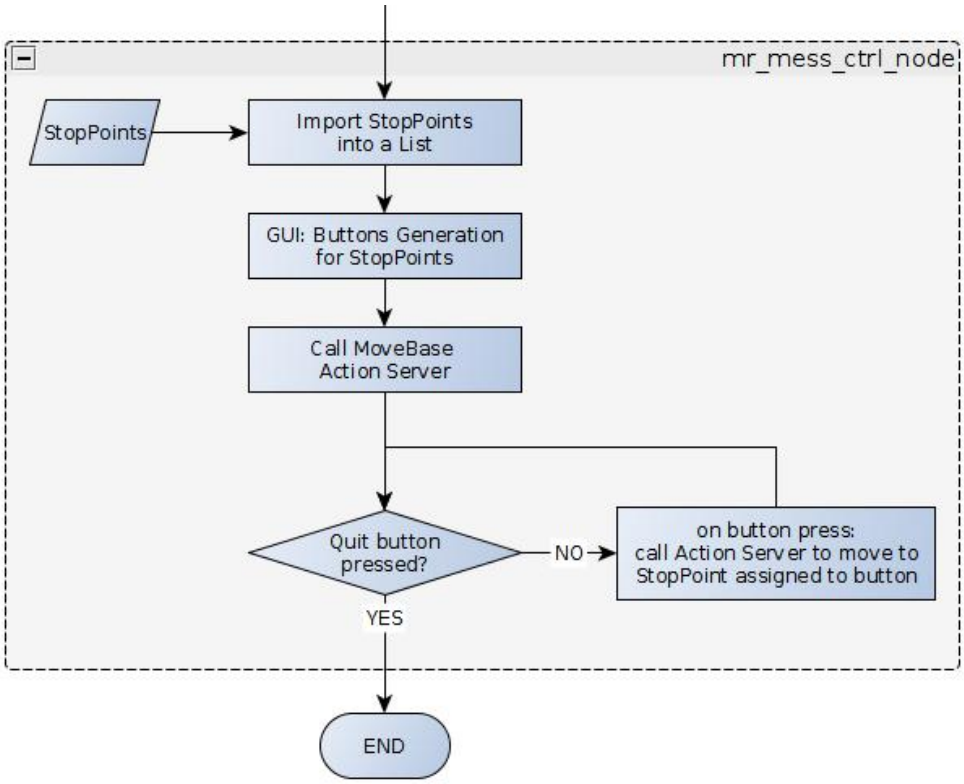
mr_mess_ctrl_node

This is the interactive control program in Python. It shows a set of buttons in a touchscreen. Each button is associated to a location in the map (Stop_Points). When the user push one of the buttons, the robot goes to that location in the map using the ROS Navigation Stack.

Inside the mr_mess.launch file it is called as:

```
<!-- Mr. Mess control launch file -->
<node pkg="mr_mess" type="mr_mess_ctrl.py" name="mr_mess_ctrl_node"
    output="screen">
</node>
```

The code is simple:



It loads the file stop_points.csv that contains data of the StopPoints: StopPoint_name, Position_X, Position_Y, Orientation_Z, Orientation_W

Then it generates one button for each of the StopPoints

It calls de MoveBase action server and in case a button is pressed, the action server is called to move the robot to the pose set for the StopPoint.

For more details see the self explanatory code

How to create the map

First thing to have for a navigation system is a map. In this section we explain how to generate it and improve it.

Basically we will have the Joystick ready and run in several terminal sessions the following commands:

```
roscore
source ~/catkin_ws/devel/setup.bash
rosparam set /robot_state_publisher/use_tf_static false
roslaunch turtlebot_bringup minimal.launch
roslaunch rplidar_ros rplidar_setup2.launch
roslaunch turtlebot_teleop logitech.launch
rosrun gmapping slam_gmapping
```

Use RViz to visualize the map building process:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Move the robot around to create the map.

Finally, save the map to disk:

```
rosrun map_server map_saver -f map
```

it will generate one .pgm and one .yaml files. The first one has the bitmap image and the second one some info about the map.

Edit and improve the map

pgm image map has a lot of noise data. The image could be improved, cropped and rotated adequately. For that we could use Gimp (or any other image editor). Here we propose some hints using Gimp (https://www.gimp.org/).

Some edits to do on map:

Rotate
Usually the map is rotated. We could try to align it with the field of view of the monitor.
Zoom the area of the map
Select Layer, Transform, Arbitrary Rotation and move the slider to rotate it accordingly.

Crop

Crop the map to the area with relevant information.

Use the Rectangle selection to select the area to crop. Then menu Image, Crop to selection.

Clear areas

There are many noisy points in areas that we know there are no obstacles.

Use the color picker tool to take the color of a clear area.

Use pencil tool and select thickness of tip. Then paint the areas not clear that should be. Is possible to do lines with shift. This helps specially to make aisles clear.

Redefine edges

Some times edges are not clearly defined in areas that should be. Use pencil tool and black color for them.

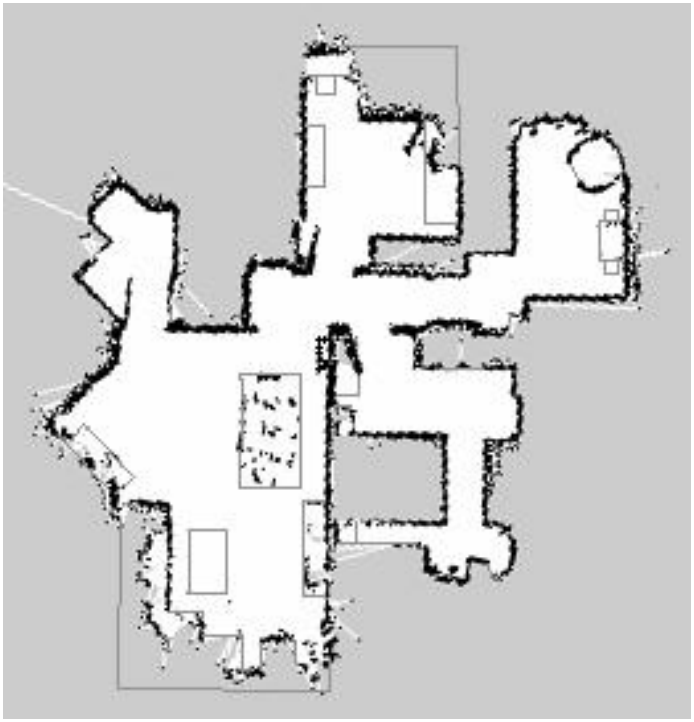
Forbidden areas

Paint a line where the robot should not go.

Bend an Image

Sometimes due to odometry errors or lack of odometry calibration, long aisle may appear bended in the maps and may be corrected by Gimp.: https://docs.gimp.org/en/plug-in-curve-bend.html

Then Save the map with Gimp format, just in case we need to do modifications in the future, and export as pgm. Ensure the extension is .pgm and saves as raw, not ascii



Now we need to modify the yaml file.

Edit the yaml file with any editor, like gedit. Modify the following data:

- √ **image**: Name of the file containing the image of the generated Map. Ensure the name of the file is the one exported from Gimp.
- √ **resolution**: Resolution of the map (in meters/pixel). Do not modify if you didn't change the size of the image. Just consider this value for the calculation of the next one.
- √ **origin**: Coordinates of the lower-left pixel in the map. This coordinates are given in 2D (x,y) in meters, If it is ok to have the origin of the coordinates of the map at the lower-left corner, just set as 0,0. The third value indicates the rotation. If there's no rotation, the value will be 0. So the steps are to look for the properties of the file, size and calculate what is mentioned before.

example of map.yaml file:

```
image: map.pgm
resolution: 0.050000
origin: [0.000000, 0.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

after editing the map, leave it here: /home/turtlebot/catkin_ws/src/mr_mess/maps/

Identify key Stop Points

Stop Points are points of destination for the robot. We will ask the robot to go to any of those points, like the kitchen, the main door, the children room, the main room or the living room in case we are using the robot in a home, or the meeting room, the laboratory, the manager's office and so on in case of using in an office area. For this case we will consider a home

We will use an excel spreadsheet to collect the poses of the Stop Points. Here is an example of such spreadsheet:

Stop_Point	Position_X	Position_Y	Orientation_Z	Orientation_W
Dock Station	6.10	9.75	0.99	0.01
Entrance	2.05	6.01	0.92	0.37
Kitchen	0.37	8.66	0.69	0.71
Living room	3.61	4.07	-0.93	0.36
Dining room	4.22	4.23	0.36	0.92
Main room	7.09	5.84	-0.88	0.46
Kids room	10.39	8.63	0.60	0.79
Study room	6.68	10.97	0.71	0.70

To get the figures we will first launch the execute the following commands in several terminal sessions:

roslaunch turtlebot_bringup minimal.launch
roslaunch rplidar_ros rplidar_setup2.launch
roslaunch turtlebot_teleop logitech.launch
roslaunch turtlebot_navigation amcl_demo.launch
map_file:=/home/turtlebot/catkin_/src/mr_mess/maps/map.yaml
roslaunch turtlebot_rviz_launchers view_navigation.launch

Use in Rviz the 2D pose Estimate button to identify the actual position and orientation of the robot.

If you have a joystick:

execute:

rostopic echo /amcl_pose

then move the robot using the joystick to the desired location. Then check for the last values shown in the terminal were you are running the rostopic echo. These are the figures you have to put in the excel spreadsheet.

If you do not have a joystick:

execute:

rostopic echo move_base_simple/goal

then use in Rviz the 2D Nav Goal button to set one by one the Stop Point poses. The robot will go to those positions. Move again the robot with this method until it is exactly located where you want. Then check for the last values shown in the terminal were you are running the rostopic echo. These are the figures you have to put in the excel spreadsheet

After filling up the excel spreadsheet, export as
/home/turtlebot/catkin_/src/mr_mess/data/stop_points.csv

Future Works

Some future improvements for this basic version of the system would be:

√ Voice feedback. Always a robot that talks is a better user experience
√ Obstacle avoidance
√ Autonomous charging. It means going autonomously to a docking station for charging when the

robot is idle or batteries are low

- √ Switch off lidar on demand. The lidar used is a low cost one. It is ok for the purpose of the project, although it is quite noisy. Switching it off when charging for example would be a good function to have
- √ Come back to previous location if it could not reach the destination and show a message explaining the issue
- √ GUI for creation of the map, configuration of stop points (using joystick), spoken messages, strategies,...

Annex 1. How to install ROS on Raspberry Pi for Turtlebot

Ubuntu Mate 16.04 installation[ii]:

Downloaded from here: https://ubuntu-mate.org/raspberry-pi/

Create SD bootable card in your computer. For that, install utilities:

sudo apt-get install gddrescue xz-utils
unxz ubuntu-mate-16.04.2-desktop-armhf-raspberry-pi.img.xz

Used disk utility to restore image to a 8GB SD card as in the video in previous link

Insert the SD card in RPi3 and boot[iii]

During the initial configuration I give name turtlebot, computer's name 'turtlebot', user name 'turtlebot', password 'turtlebot', log in automatically

Raspberry Pi Resolution

execute in a terminal in Raspberry Pi:

sudo raspi-config

then you can choose Advanced Options then Resolution, you should be able to select optimal resolution settings. After rebooting the changes should take affect.

ROS installation

Then I install ROS following this link:

https://www.intorobotics.com/how-to-install-ros-kinetic-on-raspberry-pi-3-ubuntu-mate/

The only thing that changes is the catkin directory, that I used catkin_ws instead of catkin_workspace

sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential

TURTLEBOT software

Install the Turtlebot drivers and tools software:

sudo apt-get install ros-kinetic-turtlebot-gazebo ros-kinetic-turtlebot ros-kinetic-turtlebot-apps ros-kinetic-turtlebot-interactions ros-kinetic-turtlebot-simulator

source /opt/ros/kinetic/setup.bash

Now we could connect to Turtlebot USB port and verify if it works:

roslaunch turtlebot_bringup minimal.launch
roslaunch turtlebot_teleop keyboard_teleop.launch or roslaunch turtlebot_teleop logitech.launch

and move with the joystick

RPI Lidar

Just do the following for installing the ROS drivers of RP Lidar

cd ~/catkin_ws/src
git clone https://github.com/robopeak/rplidar_ros.wiki.git
cd ..
catkin_make

Persistent USB mapping

Any time we connect or disconnect the RPLidar it is assigned to a different USB port, so lets do it persistent:
Some ideas here:http://hintshop.ludvig.co.nz/show/persistent-names-usb-serial-devices/ but summarized here:
Look for the VendorID:ProductID with Linux command lsusb with and without the Lidar connected. Compare and get the line difference, that correspond to the lidar. Example (in bold the Vendor ID:Product ID)

Bus 001 Device 020: ID **10c4:ea60** Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
After checking with command

ls -l /dev | grep ttyUSB

we get that lidar is connected to ttyUSB1:

lrwxrwxrwx 1 root root 7 Oct 29 19:22 kobuki -> ttyUSB2
crw-rw---- 1 root dialout 188, 1 Oct 29 19:22 ttyUSB1
crw-rw-rw- 1 turtlebot dialout 188, 2 Oct 29 19:32 ttyUSB2

Then we need to find out the serial number by using:
udevadm info -a -n /dev/ttyUSB1 | grep '{serial}' | head -n1

We get: ATTRS{serial}=="0001"

Go to /etc/udev/rules.d. Create a new file called 99-usb-serial.rules (sudo gedit 99-usb-serial.rules) and put the following line in there:

SUBSYSTEM=="tty", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60", ATTRS{serial}=="0001", SYMLINK+="rplidar", MODE="0666"

Now the device names will continue to be assigned ad-hoc but the symbolic links will always point to the right device node. Let’s see. Unplug rplidar and plug it back again. Then execute

ls -l /dev/rplidar

If we get the following, things are going ok:

lrwxrwxrwx 1 root root 7 Oct 29 19:45 /dev/rplidar -> ttyUSB1

Also we see:

ls -l /dev | grep ttyUSB

lrwxrwxrwx 1 root root 7 Oct 29 19:45 kobuki -> ttyUSB2
lrwxrwxrwx 1 root root 7 Oct 29 19:45 **rplidar** -> ttyUSB1
crw-rw---- 1 root dialout 188, 1 Oct 29 19:45 ttyUSB1
crw-rw-rw- 1 turtlebot dialout 188, 2 Oct 29 2017 ttyUSB2

The last step is to configure all the relevant tools to use these new names and forget about chasing the right /dev/ttyUSB* every second day.
Once you have change the USB port remap, you can change the launch file about the serial_port value.
<param name="serial_port" type="string" value="/dev/rplidar"/>

roscd rplidar_ros
cd launch
sudo gedit rplidar.launch

After editing should look like:

<launch>
<node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode" output="screen">
<param name="serial_port" type="string" value="/dev/rplidar"/>
<param name="serial_baudrate" type="int" value="115200"/>
<param name="frame_id" type="string" value="laser"/>
<param name="inverted" type="bool" value="false"/>
<param name="angle_compensate" type="bool" value="true"/>
</node>
</launch>

Publish RP Lidar frame into TF

It is necessary to publish the lidar frame referred to any other frame, An easy solution would be to manually starting a tf publisher that publishes the needed transform from the base_link (localized) to the laser scanner frame. To do that, add the following line to the file mentioned above ~/catkin_ws/src/rplidar_ros-master/launch/rplidar.launch

<node pkg="tf" type="static_transform_publisher" name="base_to_laser_broadcaster" args="0 0 0 0 0 0 base_link laser 100" />

Args are x y z orient. For an orientation of 180 degrees it is pi radians, so 3.14 (args="0 0 0 3.14 0 0)

SSH

Activate SSH on RPi by executing sudo service ssh start or raspi-config for permanent ssh activation
Then enter from another computer by using ssh turtlebot@turtlebot.local

Now configure SSH connections based on public keys:

First, generate public key and private key on local PC. Open a terminal and execute:

$ ssh-keygen

Press “Enter” to accept the default parameters, “id_rsa.pub” and “id_rsa” will be generated in “~/.ssh” folder. Then copy “id_rsa.pub” to remote

computer:

$ scp ~/.ssh/id_rsa.pub turtlebot@ip_address:/home/turtlebot (“ip_addreee” is the IP address of remote computer)

After copy “id_rsa.pub” to remote computer, login on remote computer via SSH:

$ ssh turtlebot@ip_address

After login,append the content of “id_rsa.pub” to “~/.ssh/authorized_keys” on remote computer,and change the permission of “authorized_keys” file:

```
$ mkdir -p ~/.ssh/
$ cat id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
```

Now you can login on remote computer without a password.

AUTOLOGIN

To configure the Desktop to auto-login add an autologin-user line specifying your user name to the /usr/share/lightdm/lightdm.conf.d/60-lightdm-gtk-greeter.conf file.

sudo /usr/share/lightdm/lightdm.conf.d/60-lightdm-gtk-greeter.conf
add: [SeatDefaults] greeter-session=lightdm-gtk-greeter autologin-user=turtlebot

Conf files in the /usr/share/lightdm/lightdm.conf.d/ directory are cascaded into the lightdm login manager in alpha order.

Redirect Audio Output

The sound will output to HDMI by default if both HDMI and the 3.5mm audio jack are connected. You can, however, force the system to output to a particular device using raspi-config.

For those of you who want to know how to do this without raspi-config:

For HDMI

sudo amixer cset numid=3 2

For 3.5mm audio jack

sudo amixer cset numid=3 1

Remote desktop

Two options: Share the existing X active desktop or create virtual instances. We need to install a VNC server and access it via Remmina on VNCviewer

Virtual instances:

Use vnc4server:
Follow instructions
here: http://www.linuxeveryday.com/2017/08/install-configure-vnc-server-ubuntu-mate
Add to the xstartup configuration file mentioned in the link the following two lines:
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS

Remote access to active desktop

We use teamviewer.
Access in RPi this link to download the server
app: https://download.teamviewer.com/download/linux/teamviewer-host_armhf.deb
cd Downloads
chmod +x teamviewer-host_armhf.deb
sudo dpkg -i teamviewer-host_armhf.deb
there will be several unmet dependencies. To fix this first type “sudo apt-get update”
“sudo apt-get -f install” to install all the dependencies or use “sudo apt-get -f upgrade” to install dependencies as well as upgrade other modules.
reboot
By default, the Teamviewer will start at the boot

Touchscreen display

If you get the official Raspberry Pi 7” touchscreen used in this project, you could install it using this tutorial: https://youtu.be/tK-w-wDvRTg

If after installing you see the screen inverted, just do the following:

open /boot/config.txt in your favourite editor and add the line:

lcd_rotate=2

This will rotate both the LCD and the touch coordinates back to the right rotation for our display stand.

Don't use the documented display_rotate, it performs a performance expensive rotation of the screen and does not rotate the touch input.

On-screen keyboard

Ubuntu Mate already comes with an on-screen keyboard. To get it go to the menu /Aplications /Universal Access /OnBoard

Backup/Restore SD card

It is good to keep a backup of the SD card and eventually restore it if needed. For than, follow this instructions if you have a Linux computer[iv]:

Before inserting the SD card into the reader on your Linux PC, run the following command to find out which devices are currently available:

```
df -h
```

Which will return something like this:

```
Filesystem 1K-blocks Used Available Use% Mounted on
rootfs 29834204 15679020 12892692 55% /
/dev/root 29834204 15679020 12892692 55% /
devtmpfs 437856 0 437856 0% /dev
tmpfs 88432 284 88148 1% /run
tmpfs 5120 0 5120 0% /run/lock
tmpfs 176860 0 176860 0% /run/shm
/dev/mmcblk0p1 57288 14752 42536 26% /boot
```

Insert the SD card into a card reader and use the same df -h command to find out what is now available:

```
Filesystem 1K-blocks Used Available Use% Mounted on
rootfs 29834204 15679020 12892692 55% /
/dev/root 29834204 15679020 12892692 55% /
devtmpfs 437856 0 437856 0% /dev
tmpfs 88432 284 88148 1% /run
tmpfs 5120 0 5120 0% /run/lock
tmpfs 176860 0 176860 0% /run/shm
/dev/mmcblk0p1 57288 14752 42536 26% /boot
/dev/sda5 57288 9920 47368 18% /media/boot
/dev/sda6 6420000 2549088 3526652 42% /media/41cd5baa-7a62-4706-b8e8-02c43ccee8d9
```

The new device that wasn't there last time is your SD card.

The left column gives the device name of your SD card, and will look like '/dev/mmcblk0p1' or '/dev/sdb1'. The last part ('p1' or '1') is the partition number, but you want to use the whole SD card,

so you need to remove that part from the name leaving '/dev/mmcblk0' or '/dev/sdb' as the disk you want to read from.

Open a terminal window and use the following to backup your SD card:

```
sudo dd if=/dev/sdb of=~/SDCardBackup.img
```

As on the Mac, the dd command does not show any feedback so you just need to wait until the command prompt re-appears.

To restore the image, do exactly the same again to discover which device is your SD card. As with the Mac, you need to unmount it first, but this time you need to use the partition number as well (the 'p1' or '1' after the device name). If there is more than one partition on the device, you will need to repeat the umount command for all partition numbers. For example, if the df -h shows that there are two partitions on the SD card, you will need to unmount both of them:

```
sudo umount /dev/sdb1
sudo umount /dev/sdb2
```

Now you are able to write the original image to the SD drive:

```
sudo dd bs=4M if=~/SDCardBackup.img of=/dev/sdb
```

The bs=4M option sets the 'block size' on the SD card to 4Meg. If you get any warnings, then change this to 1M instead, but that will take a little longer to write.

Again, wait while it completes. Before ejecting the SD card, make sure that your Linux PC has completed writing to it using the command:

```
sudo sync
```

Some documentation

Intel Euclid integration:

- √ Intel Euclid - Turtlebot Follower: https://communities.intel.com/docs/DOC-111941
- √ Clearpath docs: https://www.clearpathrobotics.com/2018/03/clearpath-robotics-open-sources-turtlebot-euclid-design/

Power Supply:

- √ Power supply for Raspberry pi. Safely switch on and off using one button:
- √ https://www.audiophonics.fr/en/raspberry-pi-and-other-sbc-accessories/audiophonics-pi-spc-ii-power-management-power-supply-for-raspberry-pi-p-11504.html

- √ https://www.audiophonics.fr/fr/kits-modules-diy/module-de-gestion-alimentation-atx-pour-raspberry-pi-p-10912.html

---

[i] Kobuki user guide: https://docs.google.com/document/d/15k7UBnYY_GPmKzQCjzRGCW-4dIP7zl_R_7tWPLM0zKI/edit

[ii] How To install ROS Kinetic on Raspberry Pi 3 (Ubuntu Mate): https://www.intorobotics.com/how-to-install-ros-kinetic-on-raspberry-pi-3-ubuntu-mate/

[iii] Ubuntu MATE for the Raspberry Pi 2 and Raspberry Pi 3: https://ubuntu-mate.org/raspberry-pi/

[iv] Backing up and Restoring your Raspberry Pi's SD Card: https://thepihut.com/blogs/raspberry-pi-tutorials/17789160-backing-up-and-restoring-your-raspberry-pis-sd-card