



MODULE 6: DEVELOPING BASIC AUTONOMOUS VEHICLE SYSTEMS

Nicholas Ho
Institute of System Science, NUS



Contents



1. Automotive Basics
2. Step-by-Step guidelines on how to develop basic autonomous vehicle systems
3. Workshop: Develop basic autonomous vehicle systems



CHAPTER 1: **AUTOMOTIVE BASICS**





Main Parts of Automobile

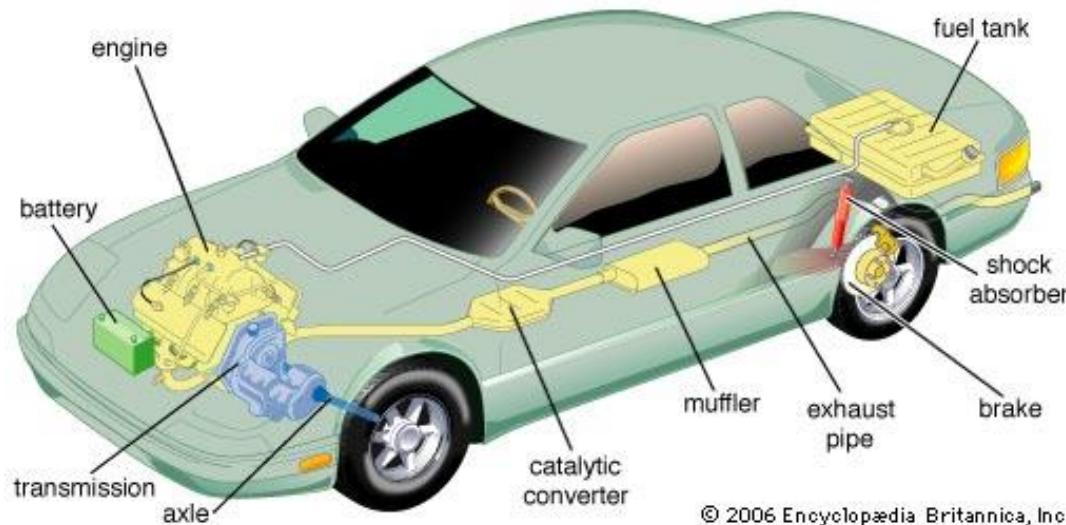


1. The Chassis

2. The Engine

3. The Transmission System

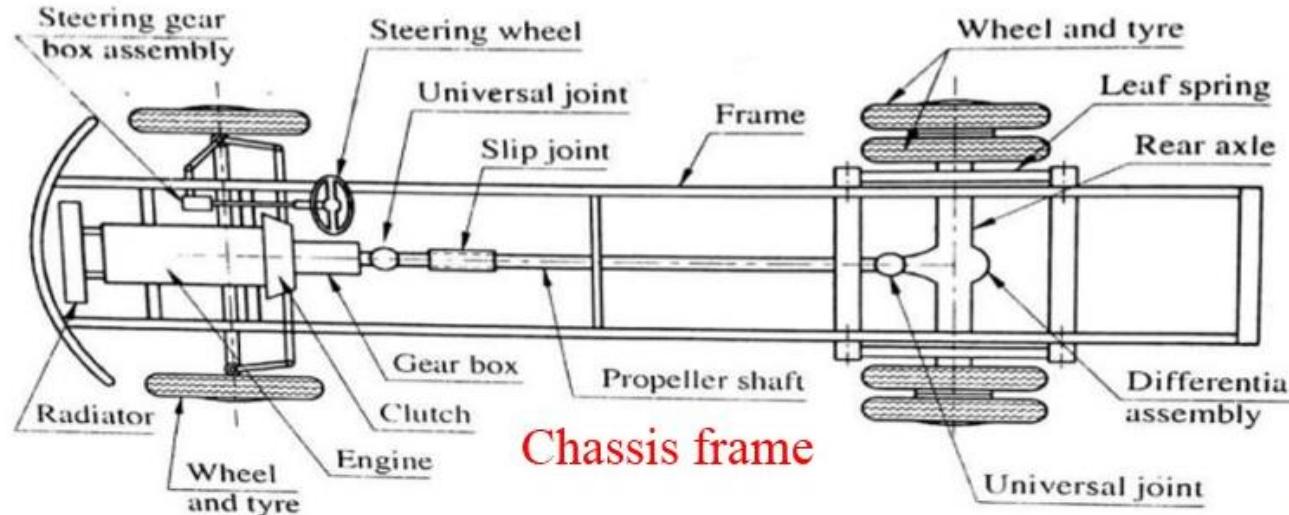
4. The Body



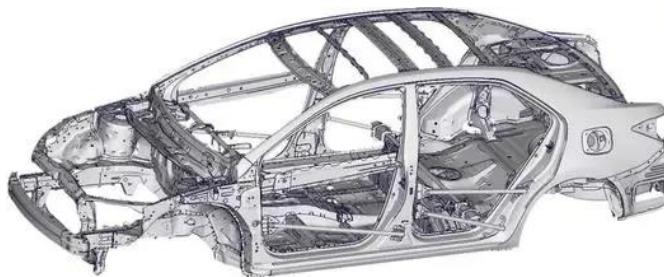
© 2006 Encyclopædia Britannica, Inc.



The Chassis



- **Main skeleton of vehicle**
- Supports engine, power transmission, and car body

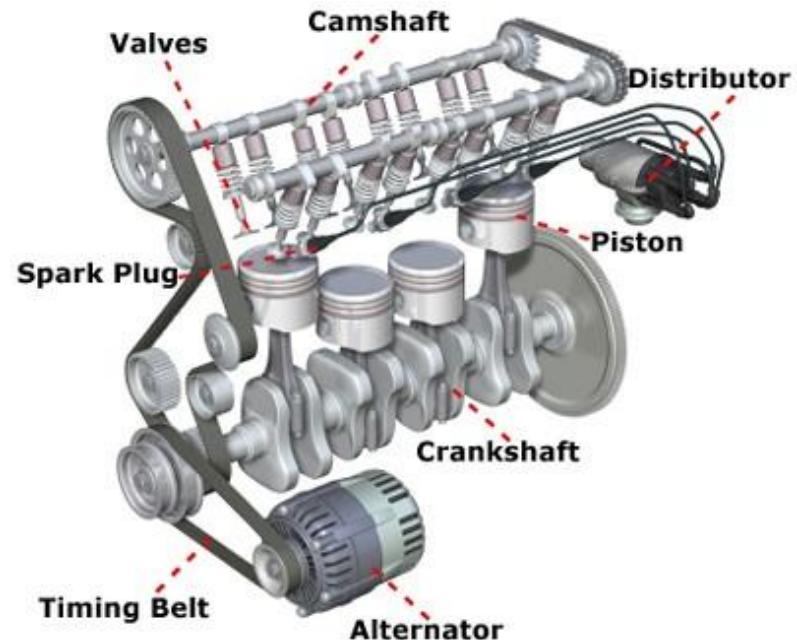




The Engine



- **Source of mechanical power to an automobile**
- E.g. internal combustion engine for general vehicles, electric motor for all-electric vehicles (EVs)

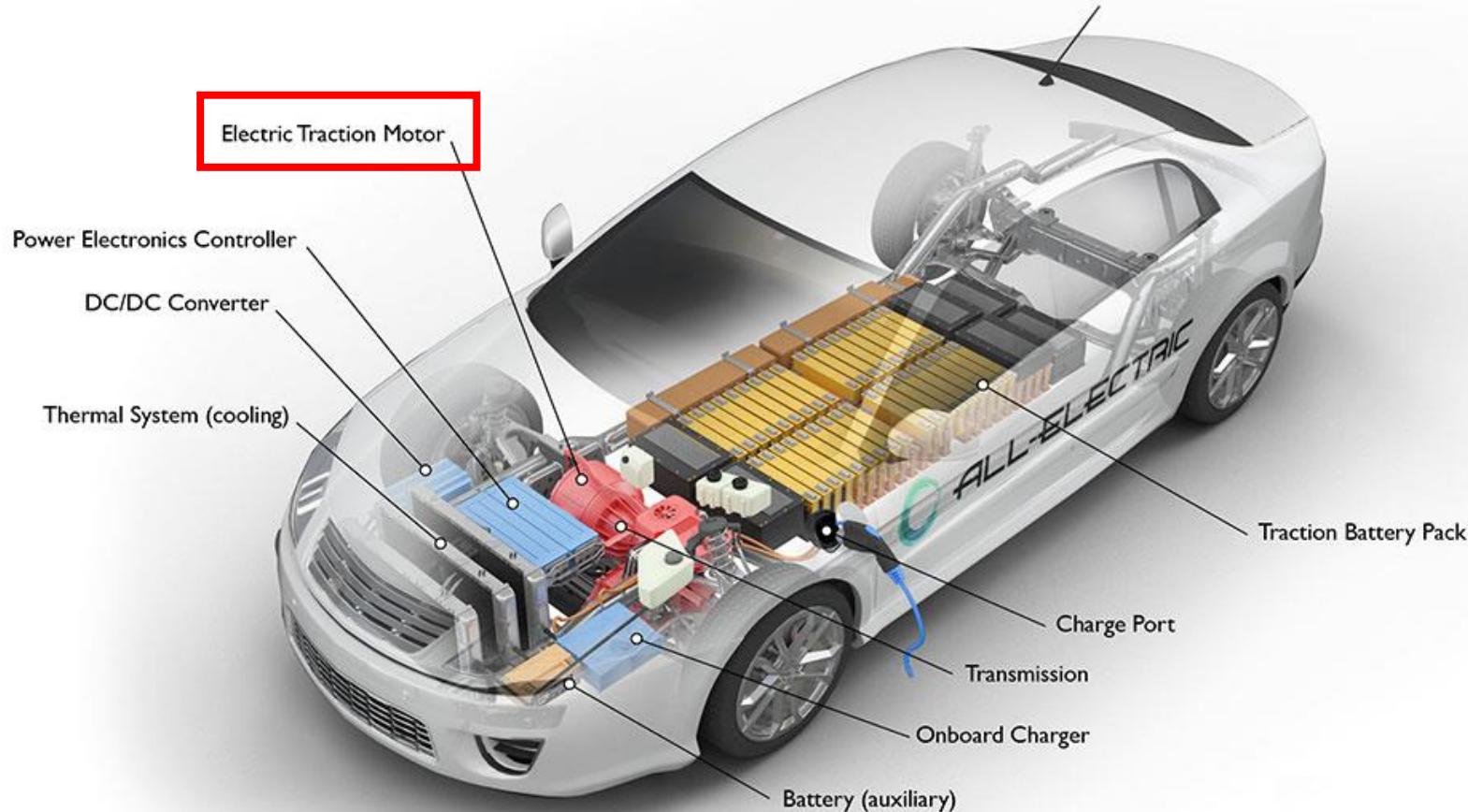




The Engine



All-Electric Vehicle



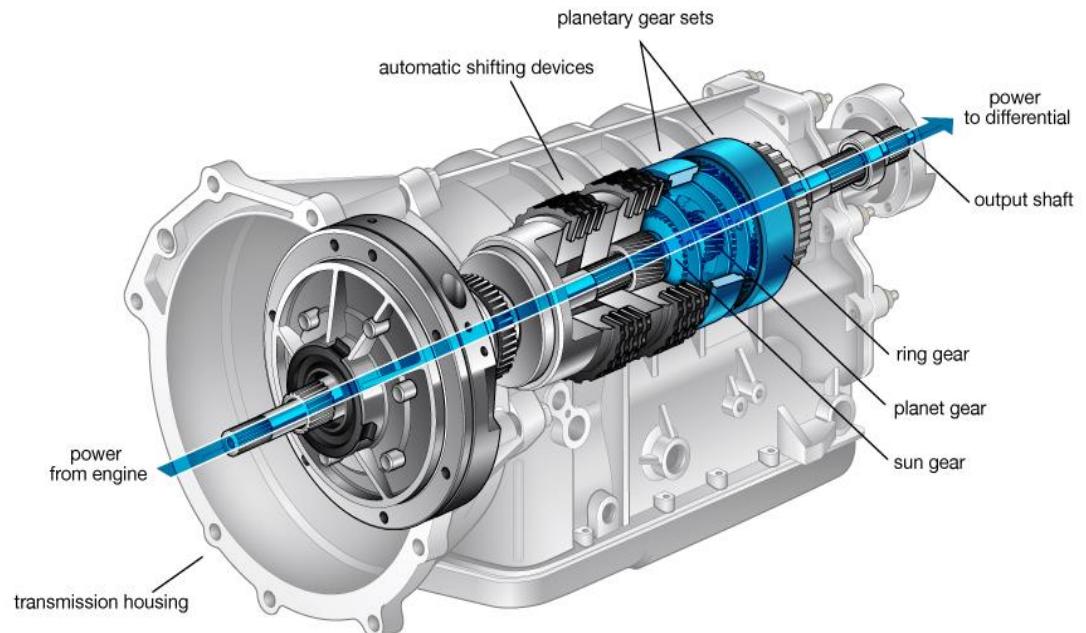
afdc.energy.gov



The Transmission System



- **Transmits power from the engine to the wheels**
- Rotation movement transferred to the wheels → leads to their rotary motion → leads to vehicle movement

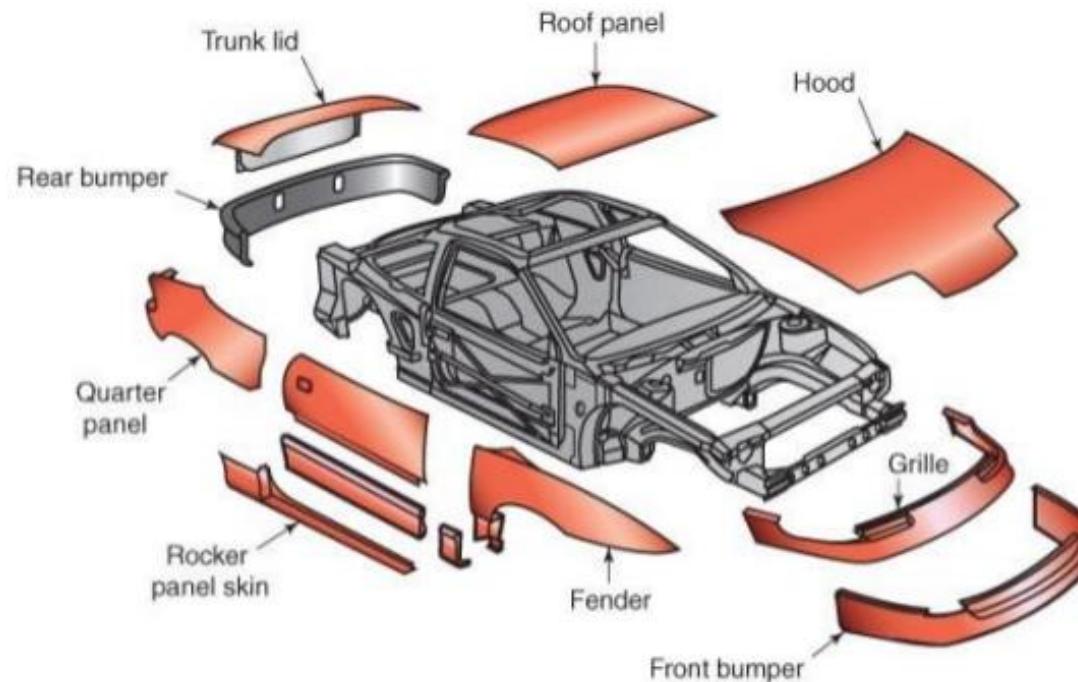




The Body



- To **protect the crucial car components** (i.e. engine, transmission system, braking/steering system) **and the passengers** from damages and/or bad weather
- Attached onto the Chassis

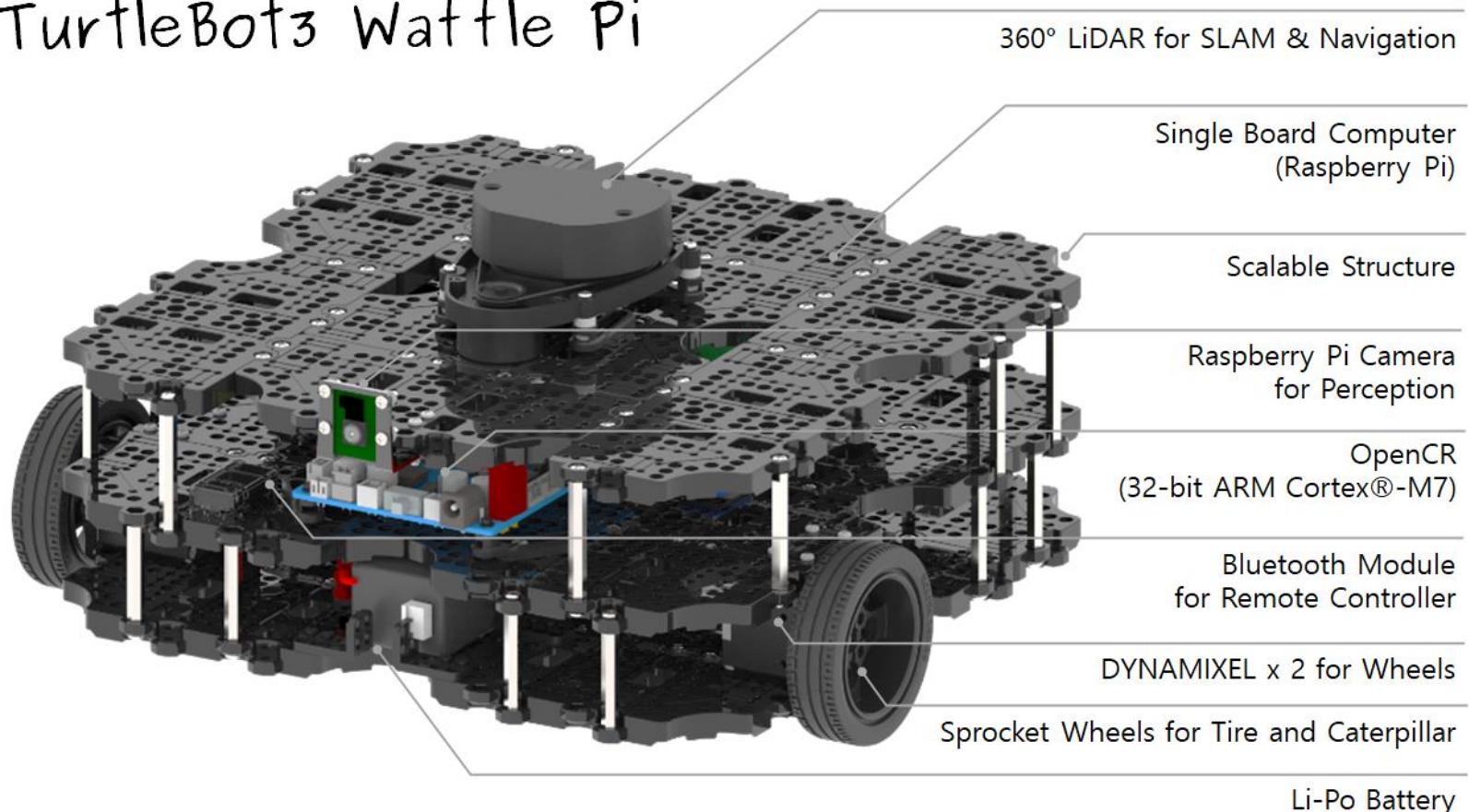




Main Parts of Automobile



TurtleBot3 Waffle Pi



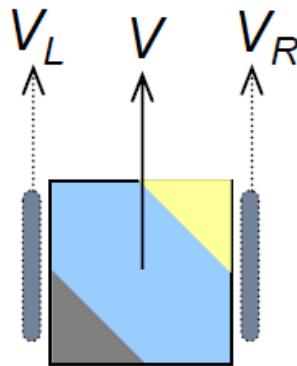
Where are the 4 main components for the TB3?



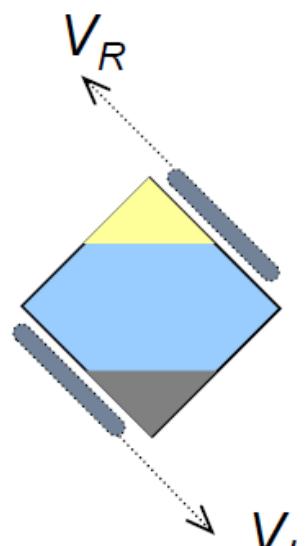
Differential Drive



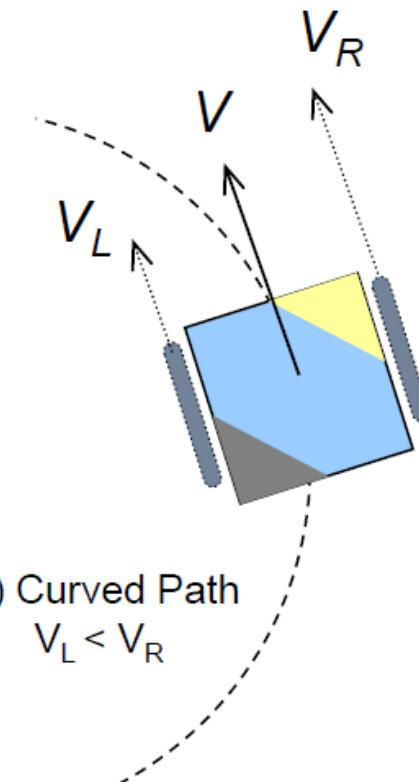
Consider the two-wheeled differential drive robot (e.g. the TB3):



(a) Straight line motion
 $V = V_L = V_R$



(b) Rotation about stationary position
 $V_L = -V_R$



(c) Curved Path
 $V_L < V_R$



Differential Drive



Advantages:

- Ease of implementation
- Simpler to control
- Ability to change orientation on the same spot

Disadvantages:

- Not as efficient as Ackerman steering or independently steered wheels when 4 or more wheels are used
 - Some of the wheels need to skid to some extent in order to complete the turn
- Unstable at high speeds



CHAPTER 2:

STEP-BY-STEP GUIDELINES ON HOW TO DEVELOP BASIC AUTONOMOUS VEHICLE SYSTEMS





What is Needed?



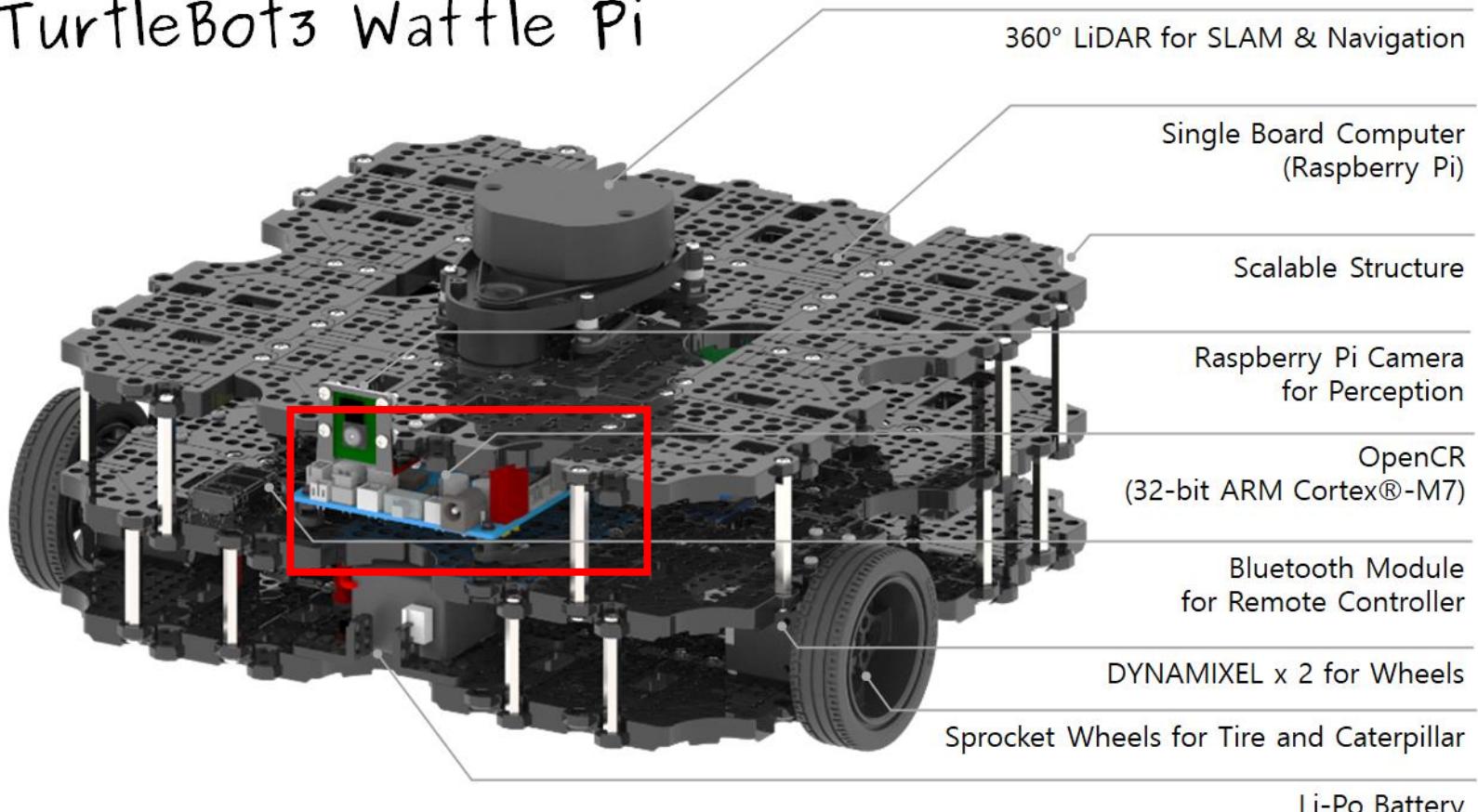
- 1. Basic automotive parts** (e.g. chassis, engine/motor, transmission system, fuel/battery)
- 2. Software** (e.g. ROS)
- 3. Embedded board; the Controller** (e.g. OpenCR)
- 4. Personal Computer** (e.g. Raspberry Pi)
- 5. Sensory Systems** (e.g. LIDAR, camera)



What is Needed?



TurtleBot3 Waffle Pi



Embedded Board for TB3 Waffle Pi:

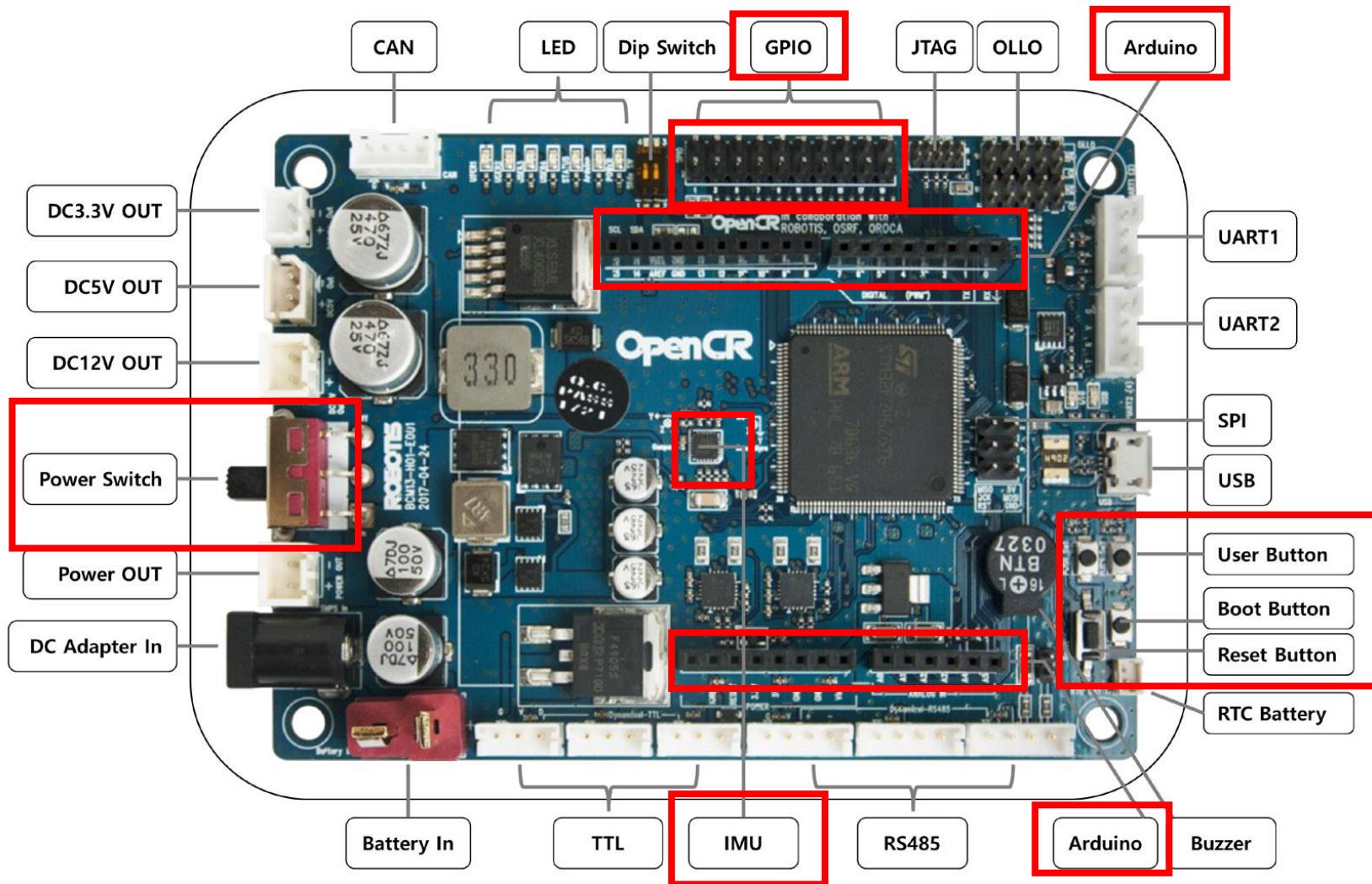
- **OpenCR**



OpenCR Interface Configuration

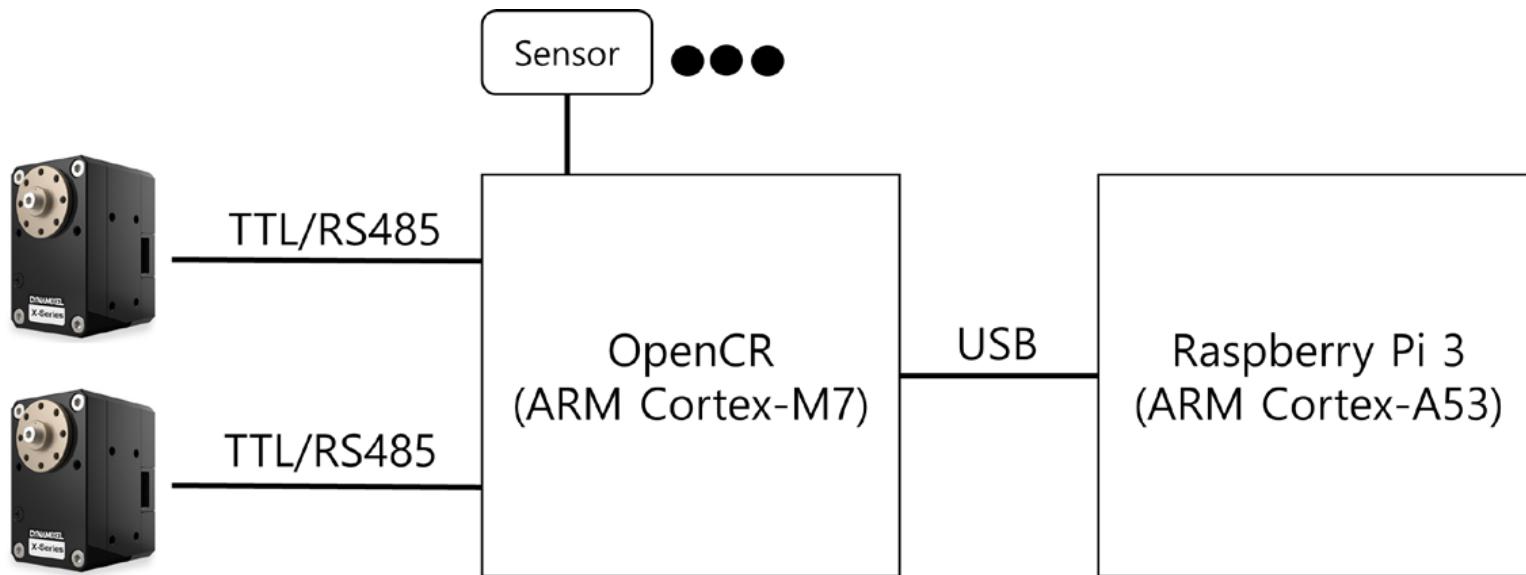








TB3 Embedded System Configuration

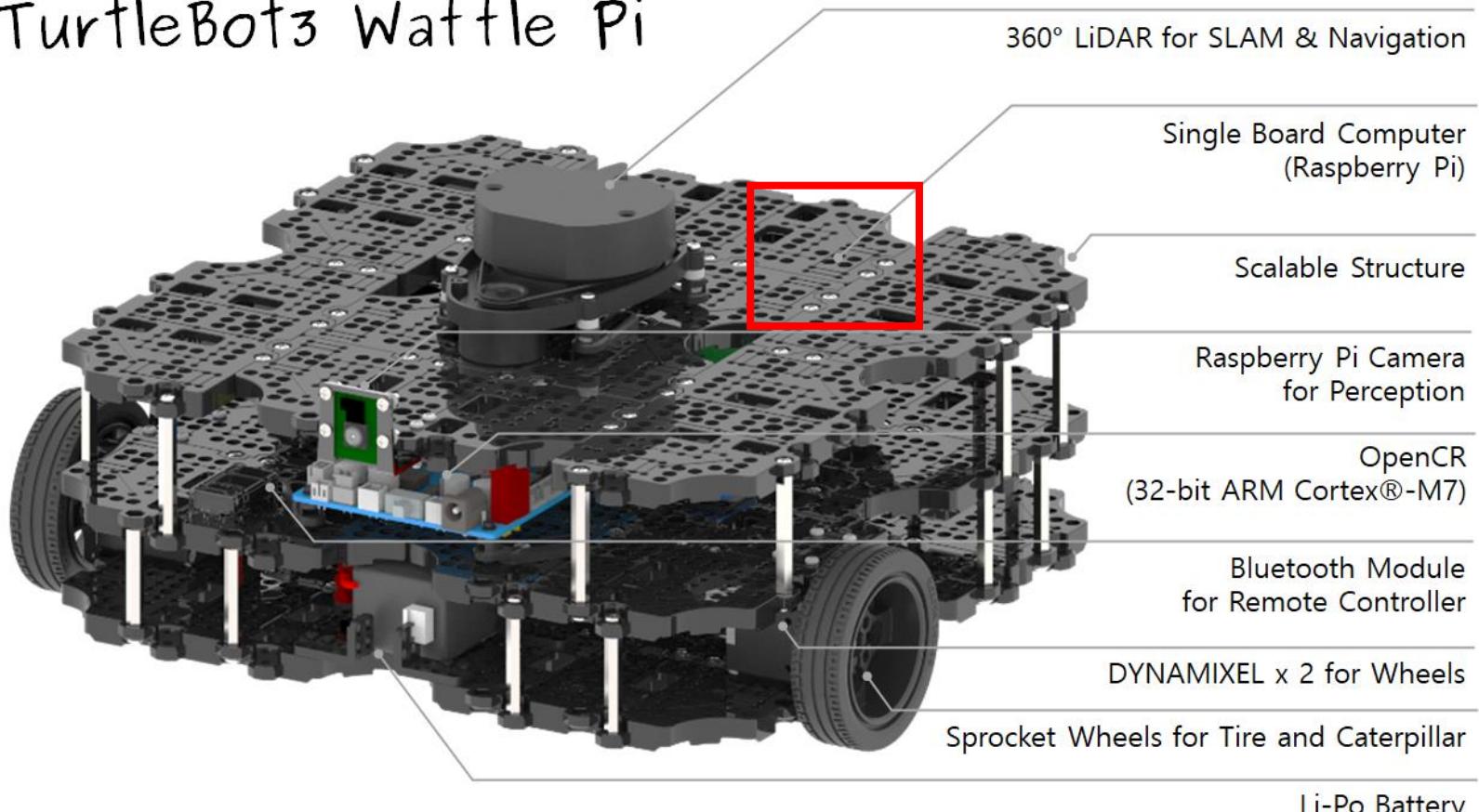




What is Needed?



TurtleBot3 Waffle Pi



PC for TB3 Waffle Pi (i.e. TB3 PC):

- Raspberry Pi



TB3 PC – Raspberry Pi



- **Main computer of the TB3**
- Enough to use the basic features of TB3 (e.g. **ROS**)
- Enable user to **communicate** with the TB3 from remote PC; connectivity between TB3 PC and remote PC

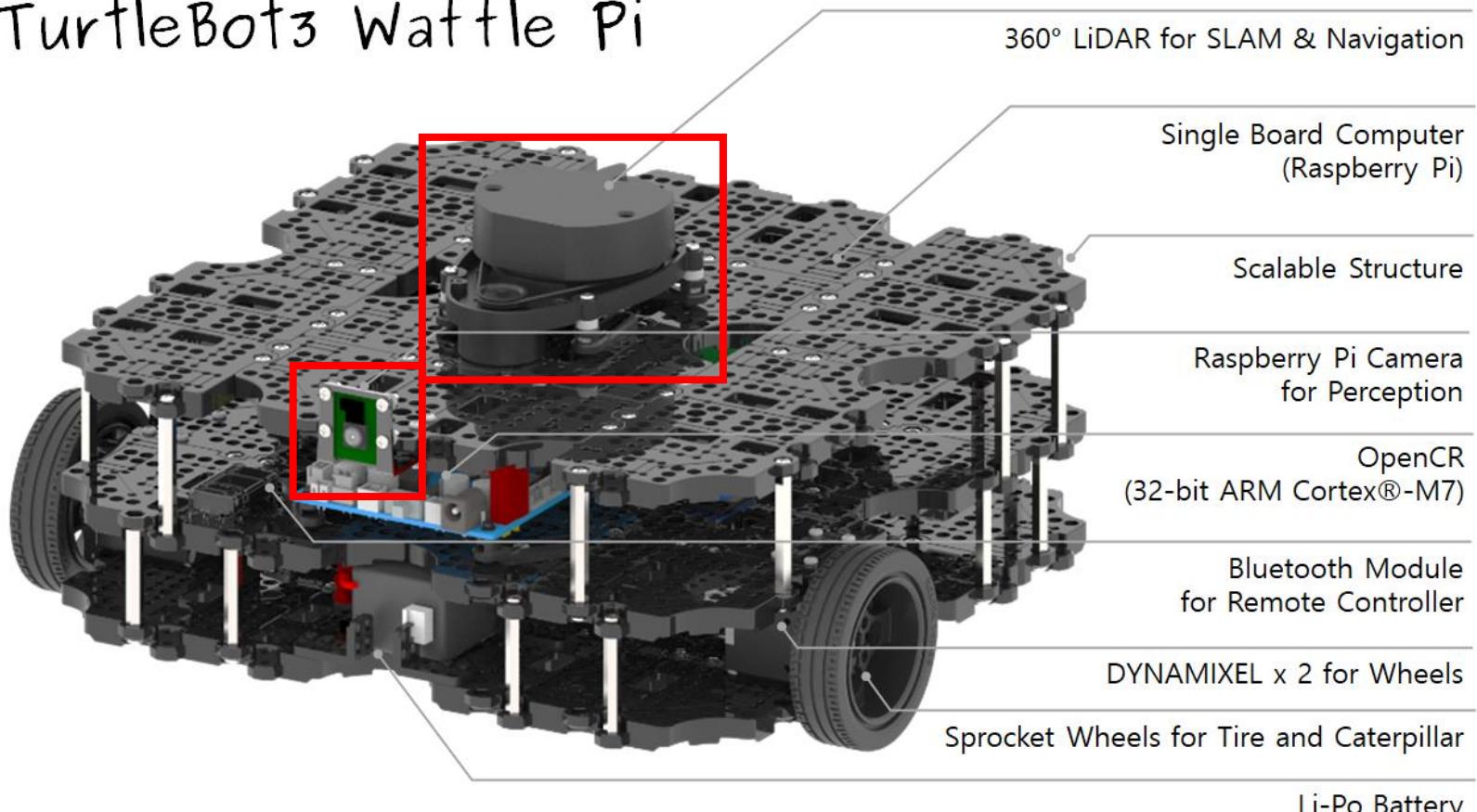




What is Needed?



TurtleBot3 Waffle Pi



Sensory Systems for TB3 Waffle Pi:
• **Camera, LIDAR**



Steps Required in Developing TB3



A. Setup Remote PC

(done; will not be extensively covered)

B. Setup TB3 PC

(done; will not be extensively covered)

C. Setup OpenCR board

(done; will not be extensively covered)

D. Setup Hardware

(done; will not be extensively covered)

P.S. For your own knowledge, these steps are detailed in the TB3 website and will not be extensively covered in this course due to time constraint. It is only for those who wants to configure the setup from scratch for their own TurtleBot3 set.



A. Setup Remote PC



Refers to Desktop/Laptop PC

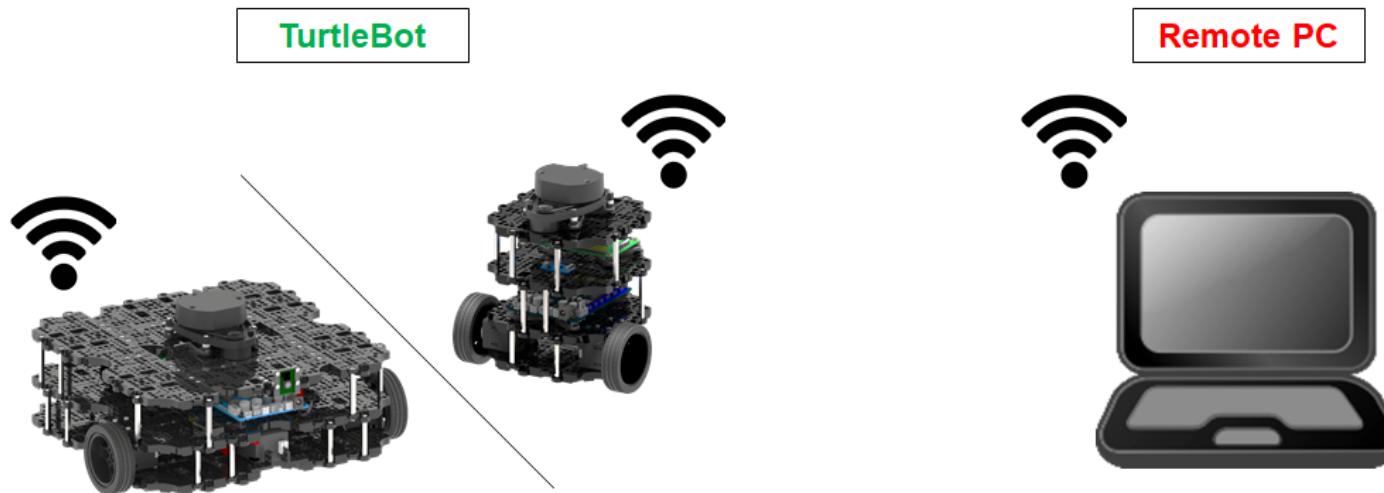
- 1. Install Ubuntu 16.04 (for physical TB3, OM and TB3_with_OM) / Ubuntu 20.04 (only for physical TB3/OM)**
- 2. Launch Ubuntu and Install ROS**
- 3. Install Dependent ROS Packages (for TB3 control on Remote PC)**
- 4. Configure Network (to enable wireless communication between TB3 PC and the Remote PC; ensure both PCs are connected to the same router)**



A. Setup Remote PC



4. Configure Network



```
ROS_MASTER_URI = http://IP\_OF\_REMOTE\_PC:11311
ROS_HOSTNAME = IP_OF_TURTLEBOT
```

```
ROS_MASTER_URI = http://IP\_OF\_REMOTE\_PC:11311
ROS_HOSTNAME = IP_OF_REMOTE_PC
```

```
$ ifconfig
```



B. Setup TB3 PC



Assuming Raspberry Pi 3 B+ is used:

- 1. Install Linux based on Raspbian/Ubuntu MATE**
- 2. Install ROS and other TB3-related packages**

P.S. There is a Linux distro image based on Raspbian, which is provided by Robotis. The image is already pre-installed with ROS and other TB3-related packages, and it supports the TB3 Burger and Waffle Pi models. This means when you transfer this image to the SD card of the Pi, you do not need to install ROS and other TB3-related packages. For more details, refer to the TB3 website



C. Setup OpenCR 1.0

1. Using a USB cable, **connect** the OpenCR board to the remote PC or to your TB3 PC
2. **Upload OpenCR Firmware for TB3**
3. **Test the board** to check if the firmware is uploaded properly or if the board is in good working condition:



Note that you can only do this test after the TB3 is at least assembled up to the 2nd layer

- Connect power to the board; the Power LED will turn on
- Place the TB3 on the floor
- Press and hold PUSH SW1 for a few seconds; move forward
- Press and hold PUSH SW2 for a few seconds; rotate 180°



D. Setup Hardware



- Involves **physical assembly of the TB3**
 - Can follow assembly manual to do the assembly OR
 - The assembly videos
 - http://emanual.robotis.com/docs/en/platform/turtlebot3/hardware_setup/#hardware-setup
- Once Remote PC, TB3 PC, OpenCR board and Hardware are all setup, you are ready to configure and control your robot



WORKSHOP: DEVELOP BASIC AUTONOMOUS VEHICLE SYSTEMS





DEMO of TB3



NICF- Autonomous Robots and Vehicles (SF)



Important to know b4 we cont!

Remote PC:

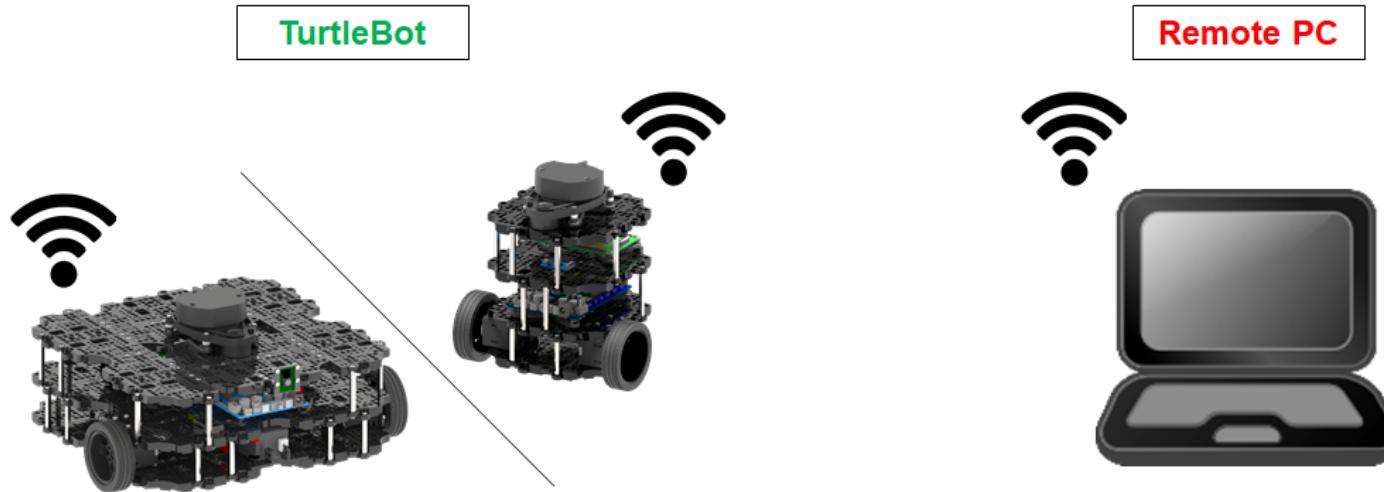


TB3 PC:





Configure Network



ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_TURTLEBOT

ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_REMOTE_PC

```
$ ifconfig
```



Configure Network



```
$ nano ~/.bashrc
```

```
alias eb='nano ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'

source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_HOSTNAME=192.168.0.100
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^J Justify
 ^T To Spell ^C Cur Pos
 ^L Go To Line

```
$ source ~/.bashrc
```



CONTROL TB3 WITH ROS



OpenCR Setup



This setup only requires you to **do it ONCE!**

1. **On the Remote PC**, open new terminal and type the following commands to access TB3 PC:

- \$ ssh ubuntu@192.168.XX.XX
- You will see that the Remote PC will log into the Raspberry Pi which is the TB3 PC

```
pi@raspberrypi: ~
nicholas@nicholas-H01987:~$ ssh pi@192.168.2.88
pi@192.168.2.88's password:
Linux raspberrypi 4.19.36-v7+ #1213 SMP Thu Apr 25 15:08:02 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Dec 25 15:32:38 2019 from 192.168.2.215
pi@raspberrypi:~ $ █
```



OpenCR Setup



This setup only requires you to **do it ONCE!**

2. **On the TB3 PC**, type the following codes in order to upload the firmware to the OpenCR

```
• $ export OPENCR_PORT=/dev/ttyACM0  
• $ export OPENCR_MODEL=waffle  
• $ rm -rf ./opencr_update.tar.bz2  
• $ wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencr_update.tar.bz2 && tar -xvf opencr_update.tar.bz2 && cd ./opencr_update && ./update.sh  
$OPENCR_PORT $OPENCR_MODEL.opencr && cd ..
```



OpenCR Setup



This setup only requires you to **do it ONCE!**

The following will appear on the terminal:

```
opencr_update/burger_turtlebot3_core.ino.bin
opencr_update/waffle_turtlebot3_core.ino.bin
opencr_update/opencr_ld_shell_arm
opencr_update/update.sh
opencr_update/
opencr_update/opencr_ld_shell_x86
opencr_update/waffle.opencr
opencr_update/burger.opencr
opencr_update/released_1.0.17.txt
armv7l
arm
OpenCR Update Start..
opencr_ld_shell ver 1.0.0
opencr_ld_main
[ ] file name      : waffle.opencr
[ ] file size       : 172 KB
[ ] fw_name         : waffle
[ ] fw_ver          : 1.0.17
[OK] Open port      : /dev/ttyACM0
[ ]
[ ] Board Name      : OpenCR R1.0
[ ] Board Ver        : 0x17020800
[ ] Board Rev        : 0x00000000
[OK] flash_erase     : 0.93s
[OK] flash_write     : 1.23s
[OK] CRC Check       : 119909C 119909C , 0.006000 sec
[OK] Download
[OK] jump_to_fw
```

When firmware upload is completed successfully, `jump_to_fw` text string will be printed on the terminal



*****Bringup*****

- The **Bringup steps below must be executed first to activate TB3 before you can execute commands** to control the TB3
- Bringup the TB3 (without Manipulator)
 1. **On the Remote PC**, run roscore (**Important: do NOT run roscore under the TB3 PC!!!**)
\$ roscore
 2. **On the Remote PC**, open new terminal and type the following commands to access TB3 PC:
\$ ssh ubuntu@192.168.XX.XX
You will see in this terminal that you are connected to the pi's terminal
- 3. **On the TB3 PC** (using the linked pi terminal), type the following to bring up basic packages to start TB3:

```
$ roslaunch turtlebot3_bringup  
turtlebot3_robot.launch
```



Other Bringup commands if you want to **launch LIDAR, Raspberry Pi Camera or core separately**:

- `$ rosrun turtlebot3_bringup turtlebot3_lidar.launch`
- **`$ rosrun turtlebot3_bringup turtlebot3_rpicamera.launch`**
- `$ rosrun turtlebot3_bringup turtlebot3_core.launch`

Note that: `$ rosrun turtlebot3_bringup turtlebot3_robot.launch` will activate the LIDAR and the core together but not the Raspberry Pi Camera, hence need to activate the camera separately



Basic Operation



1. Topic Monitor

- Helps you **monitor topics (i.e. status) of the TB3**
- E.g. battery state, velocity/position, joint states, odom, sensor states, transmission status
- **Utilizes rqt provided by ROS;** a Qt-based framework for GUI development for ROS



Basic Operation



1. Topic Monitor

- Switch on TB3 and activate Bringup first
- Ensure TB3 has enough space to move around
- On the Remote PC, run rqt
\$ rqt
- On the Remote PC, run rqt_image_view for camera view only

```
$ rqt_image_view
```



Basic Operation



1. Topic Monitor (cont); rqt example:

Topic	Type	Bandwidth	Hz	Value
▶ <input type="checkbox"/> /battery_state	sensor_msgs/msg/BatteryState			not monitored
▶ <input type="checkbox"/> /battery_state/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /cmd_vel	geometry_msgs/msg/Twist			not monitored
▶ <input type="checkbox"/> /cmd_vel/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /imu	sensor_msgs/msg/Imu			not monitored
▶ <input type="checkbox"/> /imu/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /joint_states	sensor_msgs/msg/JointState			not monitored
▶ <input type="checkbox"/> /joint_states/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /magnetic_field	sensor_msgs/msg/MagneticField			not monitored
▶ <input type="checkbox"/> /magnetic_field/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /odom	nav_msgs/msg/Odometry			not monitored
▶ <input type="checkbox"/> /odom/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /parameter_events	rcl_interfaces/msg/ParameterEvent			not monitored
▶ <input type="checkbox"/> /parameter_events/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /robot_description	std_msgs/msg/String			not monitored
▶ <input type="checkbox"/> /rosout	rcl_interfaces/msg/Log			not monitored
▶ <input type="checkbox"/> /scan	sensor_msgs/msg/LaserScan			not monitored
▶ <input type="checkbox"/> /sensor_state	turtlebot3_msgs/msg/SensorState			not monitored
▶ <input type="checkbox"/> /sensor_state/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
▶ <input type="checkbox"/> /tf	tf2_msgs/msg/TFMessage			not monitored
▶ <input type="checkbox"/> /tf_static	tf2_msgs/msg/TFMessage			not monitored
▶ <input type="checkbox"/> /tf/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored



Basic Operation



1. Topic Monitor

- Click on the checkbox to monitor the topic

Topic	Type	Bandwidth	Hz	Value
✓ /battery_state	sensor_msgs/msg/BatteryState	unknown	19.97	
✓ /battery_state/_intra	rcl_interfaces/msg/IntraProcessMessage	unknown	19.97	
□ /cmd_vel	geometry_msgs/msg/Twist			not monitored
□ /cmd_vel/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
✓ /imu	sensor_msgs/msg/Imu	unknown	19.98	
□ /imu/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
□ /joint_states	sensor_msgs/msg/JointState			not monitored
□ /joint_states/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
□ /magnetic_field	sensor_msgs/msg/MagneticField			not monitored
□ /magnetic_field/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
✓ /odom	nav_msgs/msg/Odometry	unknown	19.98	
□ /odom/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
□ /parameter_events	rcl_interfaces/msg/ParameterEvent			not monitored
□ /parameter_events/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored
□ /robot_description	std_msgs/msg/String			not monitored
□ /rosout	rcl_interfaces/msg/Log			not monitored
✓ /scan	sensor_msgs/msg/LaserScan	unknown		unknown
✓ /sensor_state	turtlebot3_msgs/msg/SensorState	unknown	19.98	
✓ /sensor_state/_intra	rcl_interfaces/msg/IntraProcessMessage	unknown	19.97	
✓ /tf	tf2_msgs/msg/TFMessage	unknown	39.95	
□ /tf_static	tf2_msgs/msg/TFMessage			not monitored
□ /tf/_intra	rcl_interfaces/msg/IntraProcessMessage			not monitored



Basic Operation



 NUS
National University
of Singapore



1. Topic Monitor

- **Expand the topic by clicking on the ► button next to the checkbox for more details**



Basic Operation



1. Topic Monitor

/battery_state → related to battery condition

Topic	Type	Bandwidth	Hz	Value
✓ /battery_state	sensor_msqs/msq/BatteryState	unknown	19.94	
capacity	float		0.0	
cell_temperature	sequence<float>		array('f')	
cell_voltage	sequence<float>		array('f')	
charge	float		0.0	
current	float		0.0	
design_capacity	float		1.7999999523162842	
header	std_msgs/Header		"	
location	string			
percentage	float		90.54999542236328	
power_supply_health	uint8		0	
power_supply_status	uint8		0	
power_supply_technology	uint8		0	
present	boolean		True	
serial_number	string		"	
temperature	float		0.0	
voltage	float		12.130000114440918	



Basic Operation



 NUS
National University
of Singapore



1. Topic Monitor

/odom → related to odometry of the TB3



Basic Operation



2. Teleoperation (aka manual operations)

Using Keyboard:

- Switch on TB3 and activate Bringup first
- Ensure TB3 has enough space to move around
- **On the Remote PC**, open new terminal and type:
`$ roslaunch turtlebot3_teleop
turtlebot3_teleop_key.launch`
- Control the TB3 using the keys w, a, s, d, x
 - w/x : increase/decrease linear velocity
 - a/d : increase/decrease angular velocity
 - space key, s : force stop
 - CTRL-C to quit



Basic Operation



2. Teleoperation (aka manual operations)

Using ROBOTIS RC100 controller (Bluetooth):

- Switch on TB3
- Ensure TB3 has enough space to move around
- Switch on the controller with the Bluetooth (master) module; the TB3 has the slave module
- Hit the buttons on the controller to move the TB3.

Note: When using RC-100, it is not necessary to execute a specific node because turtlebot_core node creates a /cmd_vel topic in the firmware directly connected to OpenCR

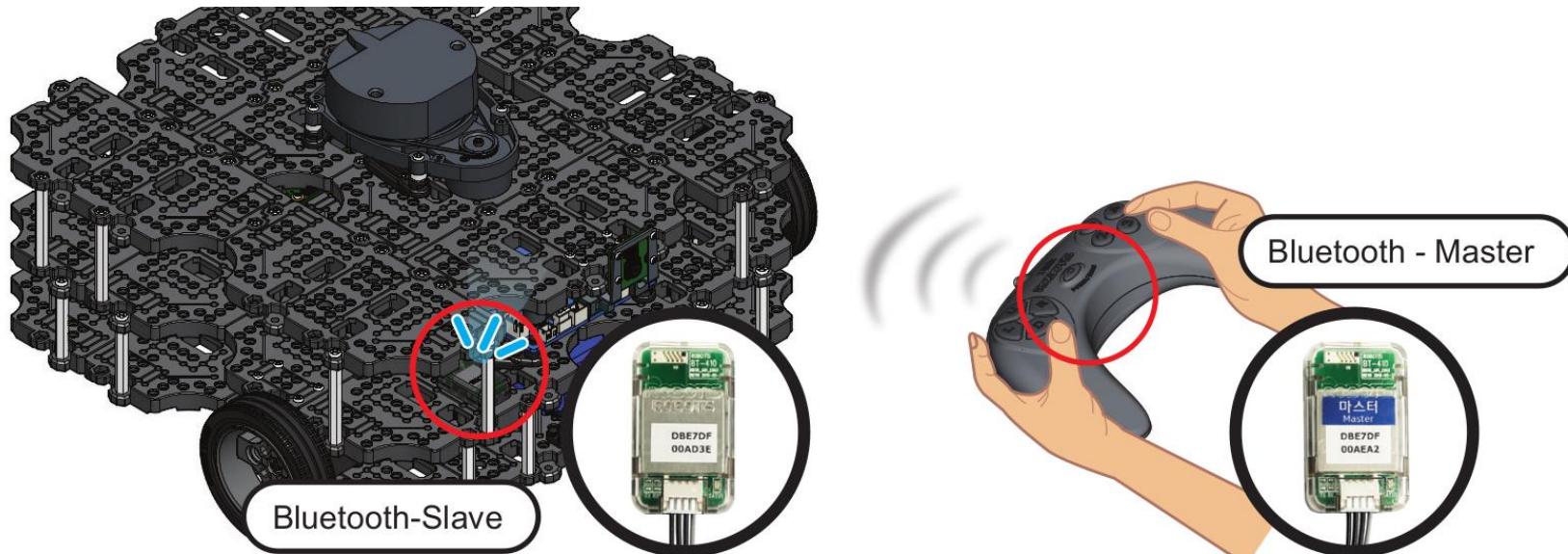


Basic Operation



2. Teleoperation (aka manual operations)

Using ROBOTIS RC100 controller (Bluetooth):



Note: When using RC-100, it is not necessary to execute a specific node because turtlebot_core node creates a /cmd_vel topic in the firmware directly connected to OpenCR



Basic Operation



3. Basic examples

Move using Interactive Markers on RViz (using the Markers to rotate or translate the TB3). **Steps:**

- Switch on TB3 and activate basic Bringup first
- Ensure TB3 has enough space to move around
- **On the Remote PC**, open new terminal and type:
`$ roslaunch turtlebot3_bringup
turtlebot3_remote.launch` **Skip this if
have errors;
not important**
- **On the Remote PC**, open new terminal and type:
`$ roslaunch turtlebot3_example
interactive_markers.launch`
- **On the Remote PC**, open new terminal and type:
`$ rosrun rviz rviz -d `rospack find
turtlebot3_example`/rviz/turtlebot3_interactive.rviz`

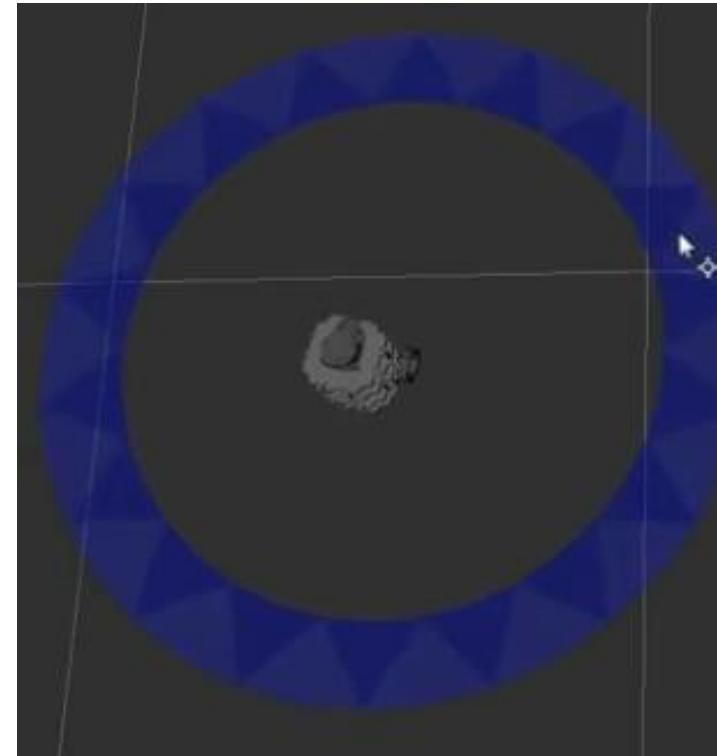
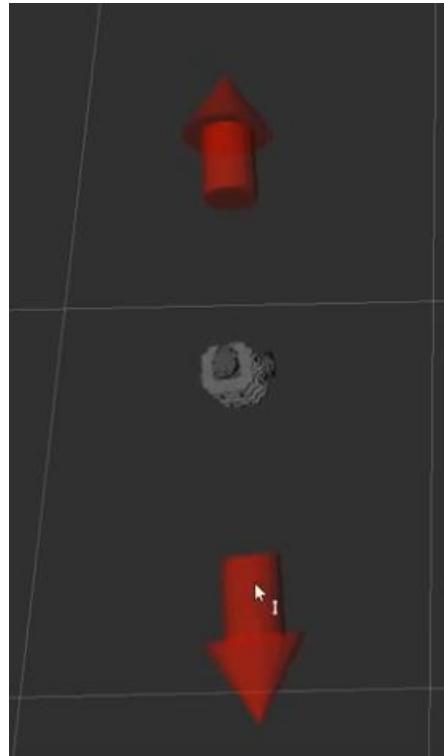


Basic Operation



3. Basic examples

Move using Interactive Markers on RViz; use the Markers to rotate or translate the TB3:





Basic Operation



3. Basic examples

Obstacle Detection (using LDS data to move or stop the TB3; stops when it detects obstacle ahead). **Steps:**

- Switch on TB3 and activate Bringup first
- Ensure TB3 has enough space to move around
- **On the Remote PC**, open new terminal and type:

```
$ roslaunch turtlebot3_example  
turtlebot3_obstacle.launch
```
- Put an obstacle in front of the TB3 to see if it stops



Basic Operation



3. Basic examples

Point Operation (move the TB3 to a point within a 2D space and rotate it); Steps:

- Switch on TB3 and activate Bringup first
- Ensure TB3 has enough space to move around
- **On the Remote PC**, open new terminal and type:

```
$ roslaunch turtlebot3_example  
turtlebot3_pointop_key.launch
```

- Type 0.0 0.0 90 in the terminal to represent points x, y and z; z is the rotational angle the TB3 will turn to



Basic Operation



4. Using Python for TB3

Try the following codes that were used in Gazebo for the previous module, M5:

1. trajectory.py **(will not work out as well as seen in simulation; why?)**
2. laser_data.py
3. avoid_obstacle.py

General Steps to execute python codes for TB3:

1. Switch on TB3 and activate Bringup first
2. Ensure TB3 has enough space to move around
3. **On the Remote PC**, open new terminal and type:

```
$ rosrun autonomous name_of_file.py
```

Steps:

- Switch on TB3 and activate Bringup first
- **Ensure TB3 has enough space to move around with ample walls/obstacles around**
- **On the Remote PC**, open new terminal and type the following commands to access TB3 PC:

```
$ ssh ubuntu@192.168.XX.XX
```

You will see in this terminal that you are connected to the pi's terminal

- **On the TB3 PC** (using the linked pi terminal), type:

```
$ roslaunch turtlebot3_bringup  
turtlebot3_rpicamera.launch
```

(optional; this will activate the Pi camera)

Steps (cont):

- **On the Remote PC**, open new terminal and type:

```
$ roslaunch turtlebot3_slam  
turtlebot3_slam.launch slam_methods:=gmapping
```

(note that the RViz tool will be automatically executed when you run the above command)

- ❖ Note that you may try the **Frontier Exploration** SLAM function **[skip this; only applicable for ROS Kinetic]** for this step too; substeps to execute this function are the same as those for the simulation module, M5 (ensure bringup is activate first):

```
$ roslaunch turtlebot3_slam  
turtlebot3_slam.launch  
slam_methods:=frontier_exploration
```

Steps (cont):

- **On the Remote PC**, open new terminal and type:

```
$ roslaunch turtlebot3_teleop  
turtlebot3_teleop_key.launch
```

- Control the TB3 using the keys w, a, s, d, x
- **Move the robot to explore the area and collect map data**; you need not explore the whole area for now (this is only for practice; you will need to explore extensively for the project later, including all corners)
- **It is important to avoid vigorous movements such as changing the speed too quickly or rotating too fast**

Steps (cont):

- On the Remote PC, open new terminal and type:

```
$ rosrun map_server map_saver -f ~/map
```
- This will save the map in the \$HOME directory
- We can use this map for Navigation purposes later



Navigation



Steps:

Part I – Run Navigation Nodes

- Switch on TB3 and activate Bringup first
- **Ensure TB3 is within the map that you have created previously**
- **On the Remote PC**, open new terminal and type:

```
$ roslaunch turtlebot3_navigation  
turtlebot3_navigation.launch  
map_file:=$HOME/map.yaml
```

(note that the RViz tool is automatically executed when you run the above command)



Navigation



Steps (cont):

Part II – Estimate Initial Pose

- In the RViz application, click on “2D Pose Estimate” button
- Click on the exact location of the actual robot position in the RViz map AND
- While holding down the left mouse button, drag the green arrow to the direction where the TB3’s front is facing (same as Gazebo example)
- This step helps to calibrate the position and direction of the TB3 within the virtual map space



Navigation



Steps (cont):

Part III – Send Navigation Goal

- **In the RViz application, click on “2D Nav Goal” button;**
a very large green arrow will appear

This green arrow is a marker that can specify the destination of the robot
- Click this arrow at your desired final destination, and drag it to set your desired final orientation of the TB3 (same as Gazebo example)
- The robot will create a path to avoid obstacles to its destination based on the map. Then, the robot moves along the path.
- At this time, even if an obstacle is suddenly detected, the robot moves to the target point avoiding the obstacle



GROUP PROJECT 4

APPLICATION OF SLAM & NAVIGATION OF TB3 WITHIN THE PHYSICAL WORLD



Group Project 4



The use of LIDAR is useful within autonomous vehicles as it is able to detect obstacles for collision avoidance purposes. Its strengths include its abilities to accurately estimate the vehicle's position (localization), to work well in the dark and to efficiently create the estimated map from collected LIDAR data.

For this project, you are to **compare the pros/cons between manual-based (RC100 OR teleop node controller) vehicles and autonomous-based vehicles using LIDAR technology.**



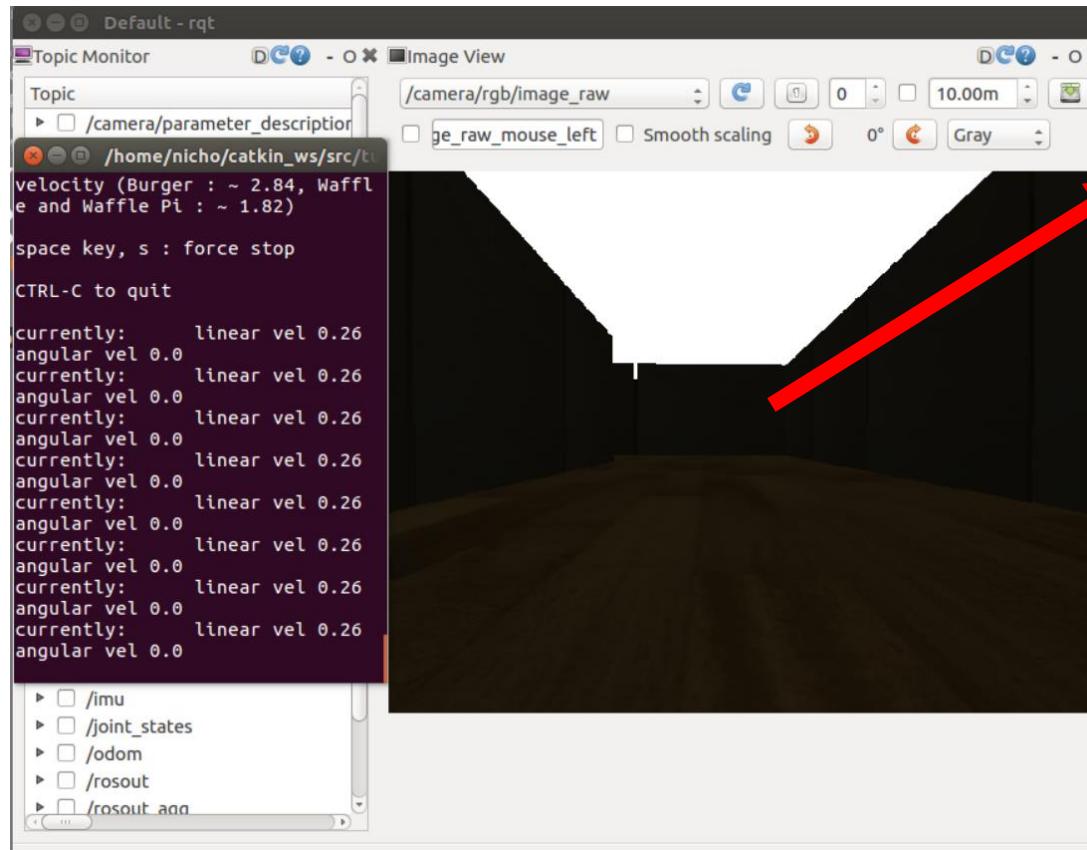
Group Project 4



In groups, you are to:

1. Firstly, try using the **RC100 OR teleop node controller** to remotely navigate the TB3 through the tunnel from the starting point to the desired destination AND orientation. You only can use the camera of the TB3 (in `rqt_image_view`) as your visual guide. Time yourself and reflect on the pros/cons of this method
2. Secondly, using the **SLAM & NAVIGATION methods** that we practiced previously, **configure an autonomous vehicle** that will automatically navigate itself within the tunnel from the starting point to the desired destination AND orientation. Use `roslaunch` and `rospy` to aid you in the navigation node (i.e. take reference from Project 3b). Time yourself and reflect on the pros/cons of this method
3. **Optional:** Put random obstacles along the paths of the vehicles during their navigation process and see what happens. Do it for both methods

Manual Driving (teleop node):



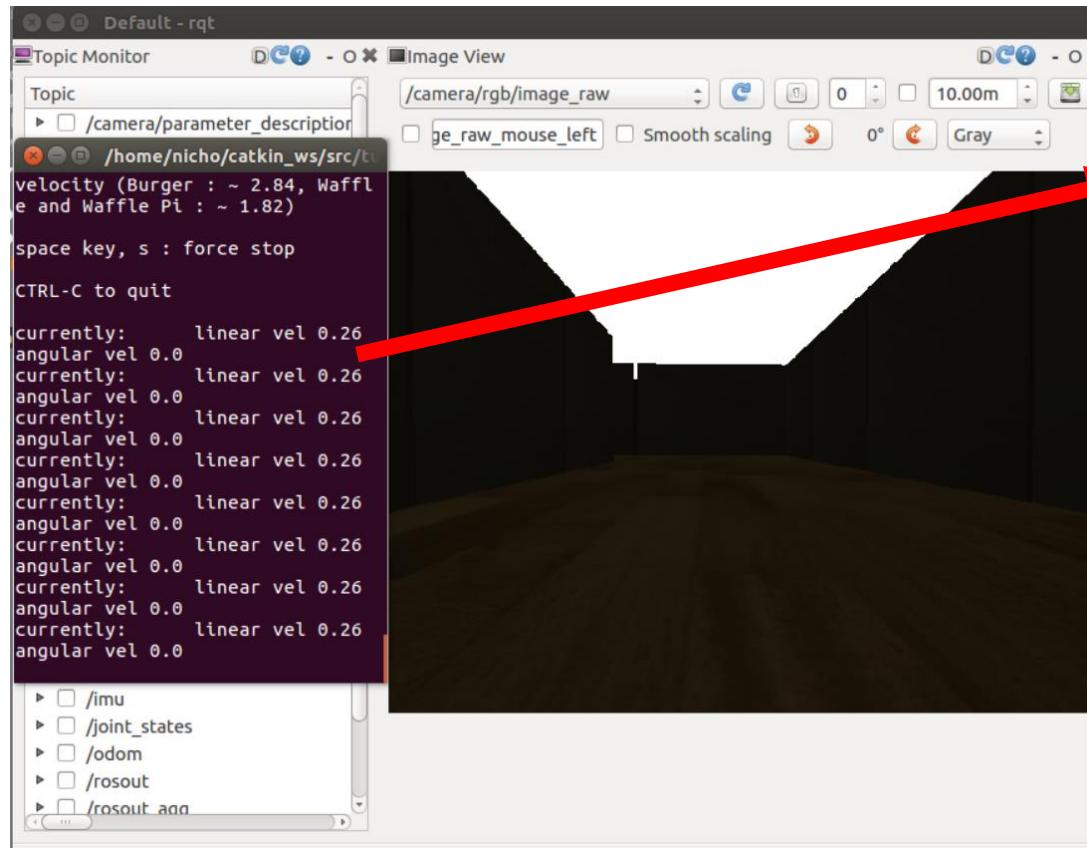
- To open up Image View, in new terminal, type `rqt_image_view`
 - Once `rqt_image_view` window is open, select the correct topic:
`/raspicam_node/image/compressed`



Group Project 4



Manual Driving (teleop node):



To activate teleop node, in new terminal, type command:

```
$ rosrun  
turtlebot3_teleop  
turtlebot3_teleop  
key.launch
```

You can use the
RC100 controller
instead of teleop



Project 4



REMINDER:

You have to **BRINGUP** your TB3 first before you execute any SLAM or NAVIGATION nodes (**refer to README for clearer and detailed instructions**); Note that when BringUp, the poses of the TB3 will all be restarted

Example for NAVIGATION node:

1. Bringup the TB3 (without Manipulator); activate the rpicamera if you are performing Manual Control
2. Execute launch file to execute required tasks:

```
$ roslaunch autonomous Project4.launch  
map_file:=$HOME/map.yaml
```

Meaning you are only required to type 1 command line to perform the given tasks



Group Project 4



IMPORTANT POINTERS:

1. **AVOID using the RC100 controller when you doing SLAM. Why???**
2. Initial pose using /odom data may not be very accurate for actual physical situations as compared to simulation. **Why???**
 - For this project, you may want to mark your initial pose. This initial pose will be where you BringUp the TB3 first before running any SLAM or navigation nodes; the /odom data will restart at origin when you BringUp the TB3
3. **Do NOT carry TB3 with your hands while the TB3 is running the BringUp node. Why???** If you accidentally do so, you have to carry the TB3 to the initial pose and restart the Bring Up node



Group Project 4



IMPORTANT POINTERS:

If the given map has narrow corners, you can **fine tune the navigation parameters** to allow your TB3 to maneuverer those narrow spaces (covered in next few slides).

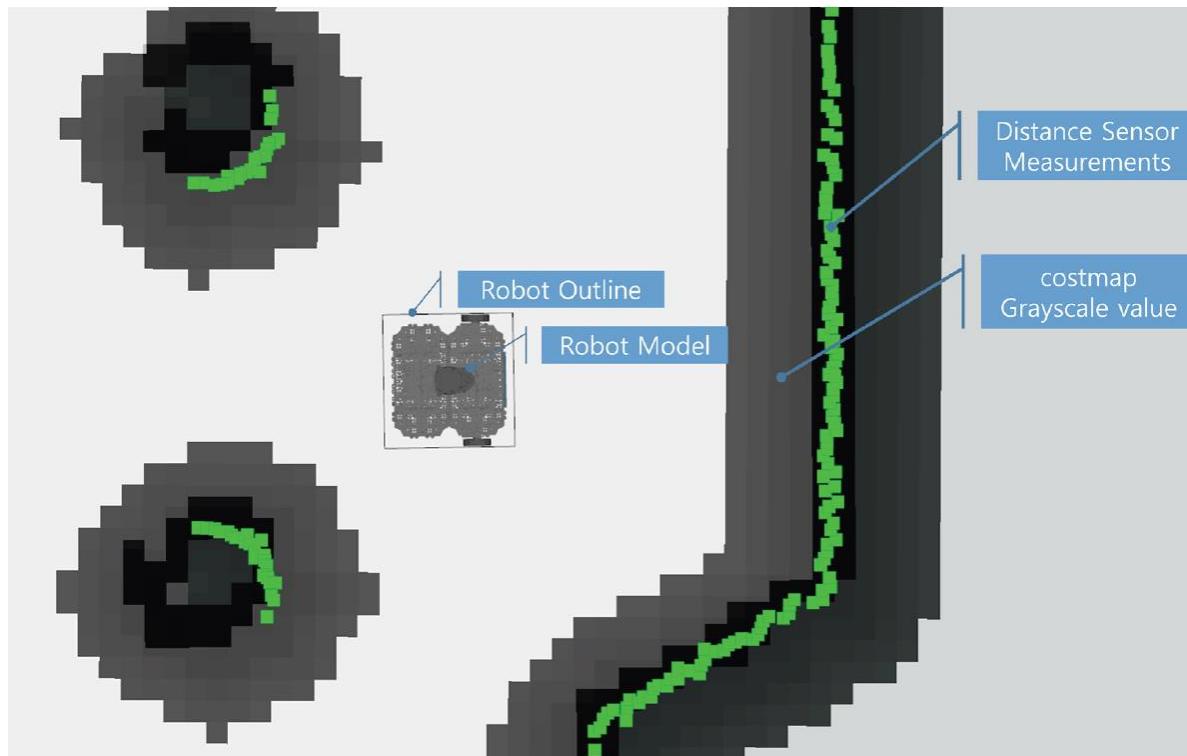
However, **for this project, you do not need to change any navigation parameters.** The **default parameters will be good enough.**



Group Project 4



Tuning Guide for Navigation Parameters (Costmap):



You can view the costmap in your RViz application when running the Navigation Node



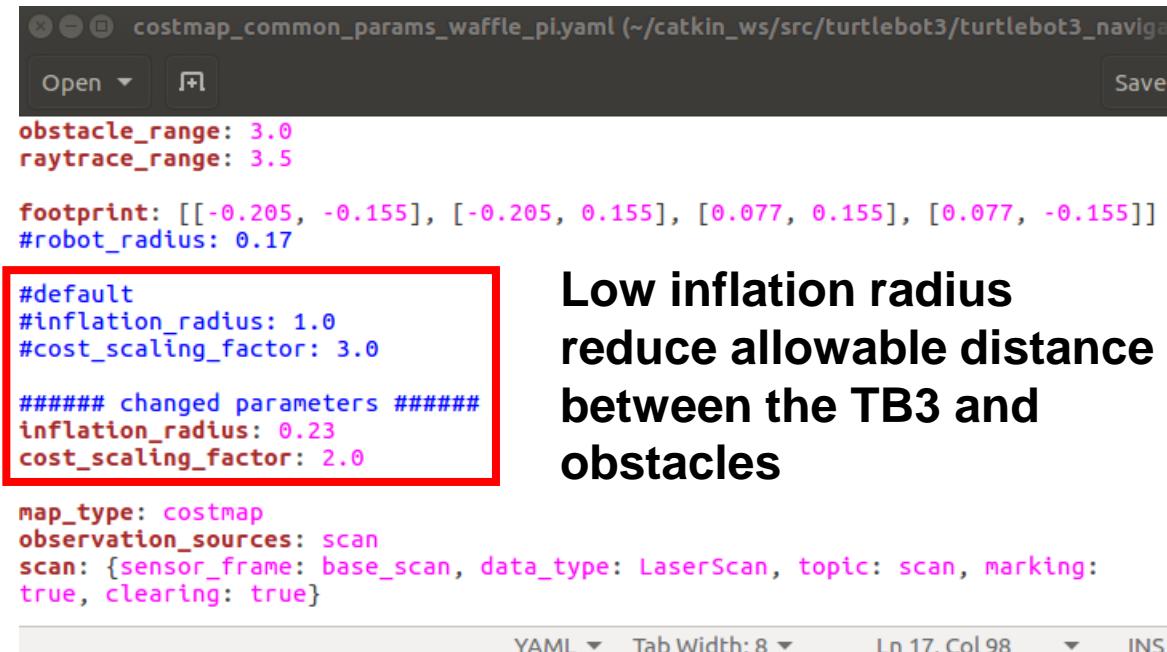
Group Project 4



Changing Costmap Parameters:

File can be found in the navigation package:

turtlebot3_navigation/param/costmap_common_params_waffle_pi.yaml



```
costmap_common_params_waffle_pi.yaml (~/catkin_ws/src/turtlebot3/turtlebot3_navigation/param)
Open Save
obstacle_range: 3.0
raytrace_range: 3.5

footprint: [[-0.205, -0.155], [-0.205, 0.155], [0.077, 0.155], [0.077, -0.155]]
#robot_radius: 0.17

#default
#inflation_radius: 1.0
#cost_scaling_factor: 3.0

##### changed parameters #####
inflation_radius: 0.23
cost_scaling_factor: 2.0

map_type: costmap
observation_sources: scan
scan: {sensor_frame: base_scan, data_type: LaserScan, topic: scan, marking: true, clearing: true}

YAML ▾ Tab Width: 8 ▾ Ln 17, Col 98 ▾ INS
```

**Low inflation radius
reduce allowable distance
between the TB3 and
obstacles**



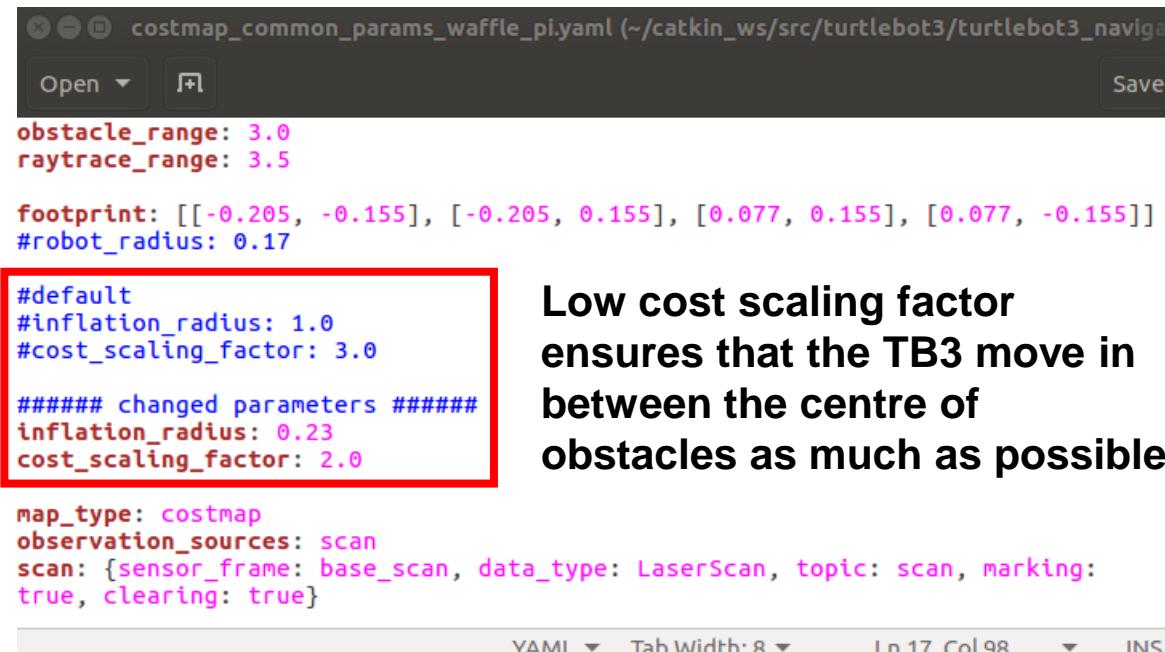
Group Project 4



Changing Costmap Parameters:

File can be found in the navigation package:

turtlebot3_navigation/param/costmap_common_params_waffle_pi.yaml



```
costmap_common_params_waffle_pi.yaml (~/catkin_ws/src/turtlebot3/turtlebot3_navigat
Open Save
obstacle_range: 3.0
raytrace_range: 3.5

footprint: [[-0.205, -0.155], [-0.205, 0.155], [0.077, 0.155], [0.077, -0.155]]
#robot_radius: 0.17

#default
#inflation_radius: 1.0
#cost_scaling_factor: 3.0

##### changed parameters #####
inflation_radius: 0.23
cost_scaling_factor: 2.0

map_type: costmap
observation_sources: scan
scan: {sensor_frame: base_scan, data_type: LaserScan, topic: scan, marking: true, clearing: true}
```

YAML ▾ Tab Width: 8 ▾ Ln 17, Col 98 ▾ INS

Low cost scaling factor ensures that the TB3 move in between the centre of obstacles as much as possible



Group Project 4



Tuning Guide for Navigation Parameters (DWA Local Planner):

- DWA = **Dynamic Window Approach**
- Popular method for **obstacle avoidance planning and avoiding obstacles**
- Method of **selecting a speed that can quickly reach a target point while avoiding obstacles that can possibly collide with the robot in the velocity search space**
- Known for its **superior performance**

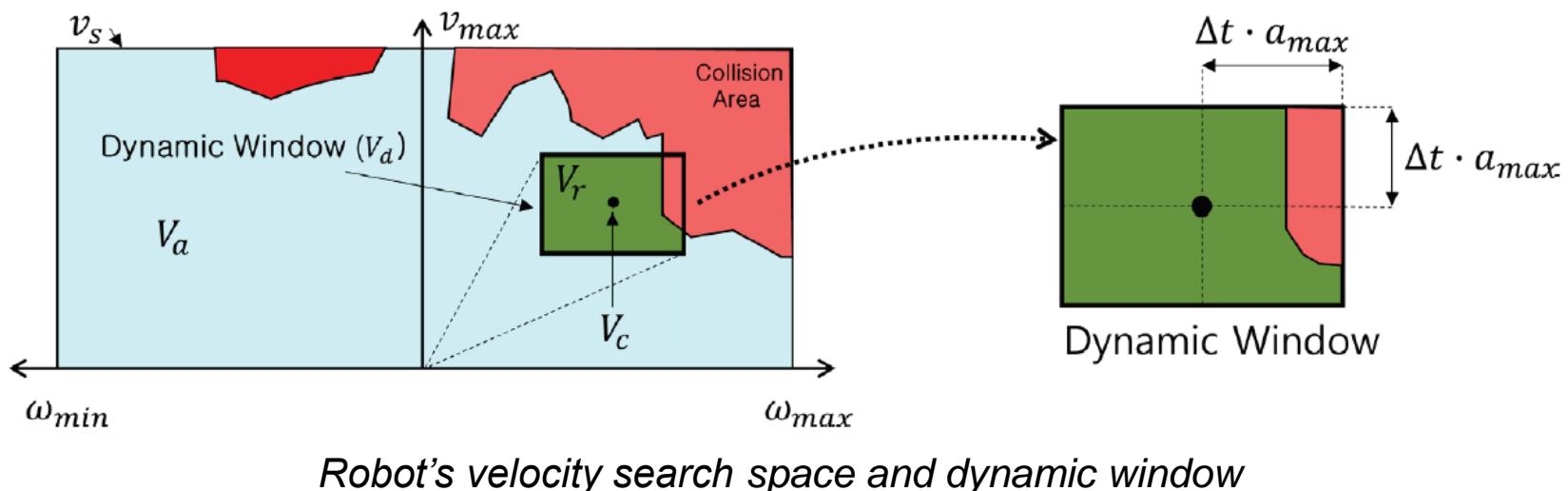


Group Project 4



DWA Theory (Summary)

Within the dynamic window, the **objective function $G(v, \omega)$** is used to **calculate the translational velocity v and the rotational velocity ω** that maximizes the objective function **which considers the direction, velocity and collision of the robot**



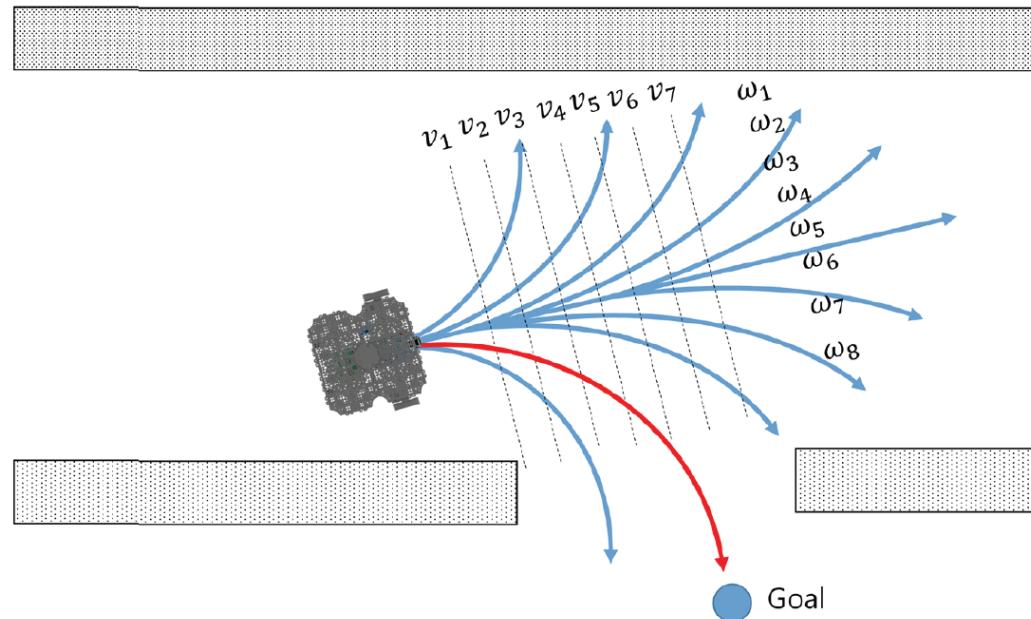


Group Project 4



DWA Theory (Summary)

The aim is to find the optimal velocities among various v and ω options to the destination



Translational velocity v and rotational velocity ω



Group Project 4



Tuning Guide for Navigation Parameters (DWA Local Planner):

Trajectory Scoring Parameters

The cost function used to score each trajectory is in the following form:

```
cost =  
    path_distance_bias * (distance to path from the endpoint of the trajectory in meters)  
    + goal_distance_bias * (distance to local goal from the endpoint of the trajectory in meters)  
    + occdist_scale * (maximum obstacle cost along the trajectory in obstacle cost (0-254))
```

~<name>/path_distance_bias

The weighting for how much the controller should stay close to the path it was given

~<name>/goal_distance_bias

The weighting for how much the controller should attempt to reach its local goal, also controls speed

~<name>/occdist_scale

The weighting for how much the controller should attempt to avoid obstacles



Group Project 4



Tuning Guide for Navigation Parameters (DWA Local Planner):

File can be found in the navigation package:

turtlebot3_navigation/param/dwa_local_planner_params_waffle_pi.yaml

```
vy_samples: 0
vth_samples: 40
controller_frequency: 10.0

# Trajectory Scoring Parameters

#default
# path_distance_bias: 32.0
# goal_distance_bias: 20.0
# occdist_scale: 0.02

##### changed parameters #####
path_distance_bias: 1.0
goal_distance_bias: 2.0
occdist_scale: 0.01

#default
forward_point_distance: 0.325
stop_time_buffer: 0.2
```

YAML ▾ Tab Width: 8 ▾ Ln 37, Col 28 ▾ INS

New parameters will help you reduce allowances for the planned paths



Group Project 4



Fill in the comparison table below:

Comparison Metrics	Manual-based (RC100 or teleop controller) vehicles	Autonomous-based vehicles using LIDAR technology
Time taken to complete driving task		
Development time taken (if any)		
Ability to estimate position of vehicle accurately in real-time		
Ability to create virtual, estimated map in real-time		
Ability to continue driving if vision is blocked or when dark		



Group Project 4



4 Items to Submit (can zip all in one file):

1. **Project4.launch**
2. **Other supporting python file(s)**
3. **Created Map files of the tunnel (both .yaml and .pgm files)**
4. **Comparison Table (word or PDF accepted)**



TURTLEBOT3 WAFFLE PI WITH ARM

[PHYSICAL; FYI ONLY]

Note that the physical TB3_with_OM system does not work well in ROS Noetic. Hence, your practice workshops for TB3_with_OM and Project 5 will be virtual-based instead.

The next few slides are there for your own information only; I will conduct a physical demo (based on ROS Kinetic) to showcase the workshops and Project 5 instead.



Manipulation



OpenCR Setup (**do it ONCE!**):

1. **On the Remote PC**, open new terminal and type the following commands to access TB3 PC:
 - \$ ssh ubuntu@192.168.XX.XX
2. **On the TB3 PC**, type the following commands in order to upload the firmware to the OpenCR (this will factor in the Open Manipulator arm):
 - \$ export OPENCR_PORT=/dev/ttyACM0
 - \$ export OPENCR_MODEL=om_with_tb3
 - \$ rm -rf ./opencr_update.tar.bz2
 - \$ wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencr_update.tar.bz2
 - \$ tar -xvf opencr_update.tar.bz2
 - \$ cd ./opencr_update && ./update.sh \$OPENCR_PORT \$OPENCR_MODEL.opencr && cd ..



Manipulation



OpenCR Setup (do it ONCE!):

The following will appear on the terminal:

```
[OK] Open port          : /dev/ttyACM0
[ ]
[ ] Board Name         : OpenCR R1.0
[ ] Board Ver          : 0x17020800
[ ] Board Rev          : 0x00000000
[OK] flash_erase       : 0.93s
[OK] flash_write       : 1.23s
[OK] CRC Check         : 119909C 119909C , 0.006000 sec
[OK] Download
[OK] jump_to_fw
```

When firmware upload is completed successfully,
jump_to_fw text string will be printed on the terminal.

You will also notice that the OM will be raised up
automatically when this setup is completed.



Manipulation (TB3)



Bringup for TB3 (with OM):

1. **On the Remote PC**, run roscore (**Important: do NOT run roscore under the TB3 PC!!!**)

```
$ roscore
```

2. **On the Remote PC**, open new terminal and type the following commands to **access TB3 PC**:

- \$ ssh ubuntu@192.168.XX.XX

3. **On the TB3 PC**, run **Bringup node for TB3, and start rosserial and LDS sensor** using following command:

- \$ roslaunch turtlebot3_bringup
turtlebot3_robot.launch

4. **On the TB3 PC**, run **the raspicam node for camera functions (OPTIONAL)**

- \$ roslaunch turtlebot3_bringup
turtlebot3_rpicamera.launch



SLAM for TB3 with OM:

You may skip the SLAM and NAVIGATION sections as they have been already done previously

1. Switch on TB3 and **activate Bringup (for TB3 with OM) first**
2. Ensure TB3 has enough space to move around with ample walls/obstacles around
3. **On the Remote PC, launch SLAM node:**
 - `$ roslaunch turtlebot3_manipulation_slam slam.launch`

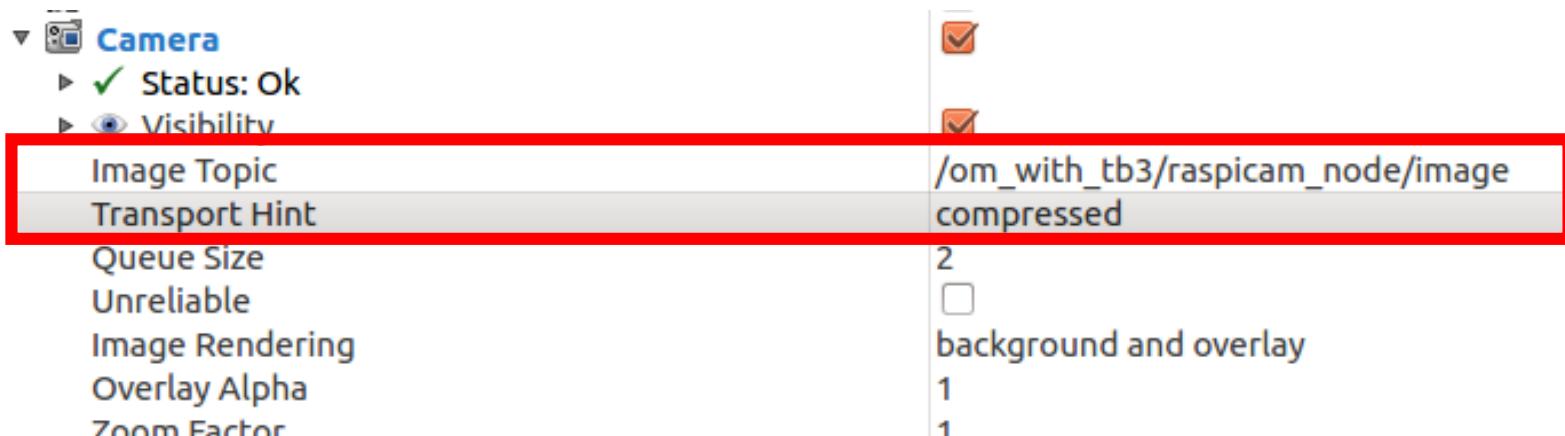


Manipulation (TB3)



SLAM for TB3 with OM:

4. On the **RViz display settings** (left side), configure the following to enable camera; under *Camera*,
 - a) Change topic name to
`/om_with_tb3/raspicam_node/image` (if this cannot work, try `/raspicam_node/image`; check `$ rostopic list` to be sure!)
 - b) Change Transport Hint to compressed



Note that for this to work, you have to run the raspicam node first



Manipulation (TB3)



SLAM for TB3 with OM:

5. On the Remote PC, launch teleop node and move around to collect map data:

- \$ roslaunch turtlebot3_teleop
turtlebot3_teleop_key.launch

6. On the Remote PC, launch map_saver node to save the explored map:

- \$ rosrun map_server map_saver -f ~/map



Manipulation (TB3)



Navigation for TB3 with OM:

1. Switch on TB3 and **activate Bringup first**
2. **Ensure TB3 is within the relevant area (which you have created a map out of previously)**
3. **On the Remote PC**, open new terminal and **launch the navigation node**:
 - `$ roslaunch turtlebot3_manipulation_navigation navigation.launch`
4. Calibrate the position and direction of the TB3 within the virtual map space in the RViz application using the “**2D Pose Estimate**” button (same as navigation for TB3)
5. Configure the desired final destination and orientation of the TB3 in the RViz application using the “**2D Nav Goal**” button (same as navigation for TB3); the OM with TB3 will respond according to these preset data



Bringup for OM (with TB3):

To move the OM while it is fixed on TB3:

1. Run roscore and the Bringup node for TB3
2. **On the Remote PC, run Bringup node for OM** on TB3:
 - \$ roslaunch turtlebot3_manipulation_bringup turtlebot3_manipulation_bringup.launch
3. **On the Remote PC, run the move_group node:**
 - \$ roslaunch turtlebot3_manipulation_moveit_config move_group.launch

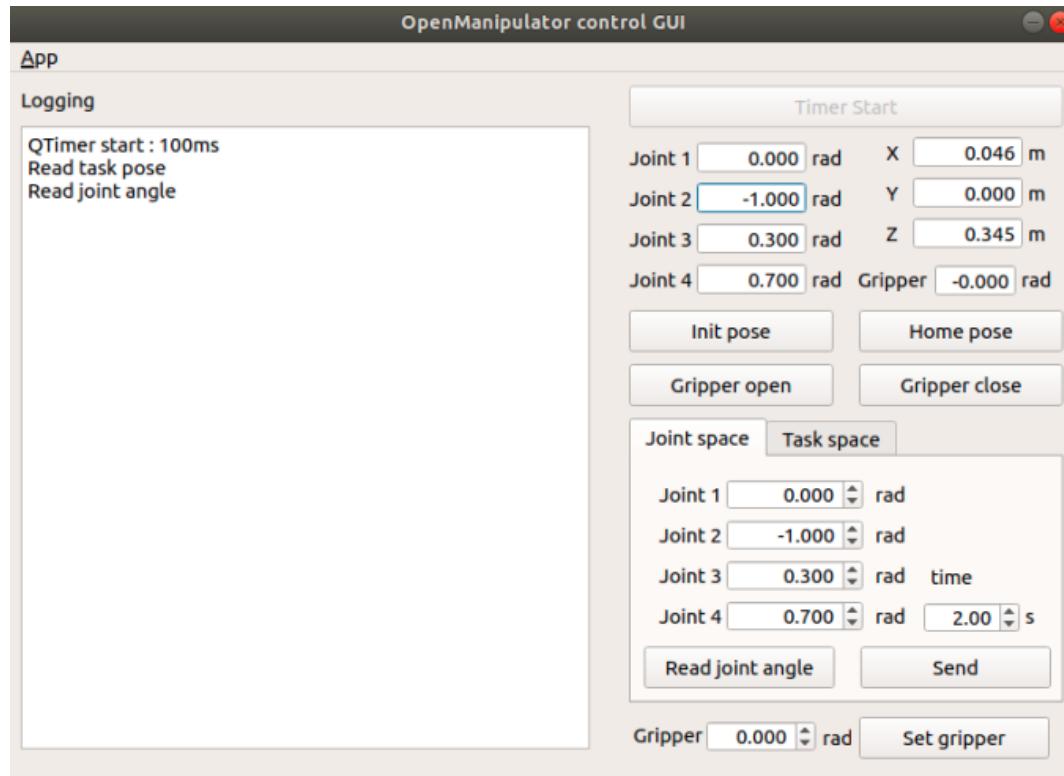


Manipulation (OM)



Using GUI: In a new terminal, type the command:

```
$ roslaunch turtlebot3_manipulation_gui  
turtlebot3_manipulation_gui.launch
```



You can **control the mounted OM using the GUI (left)**:

- **Similar to lessons taught in M3 & M5**
- **Task Space Control and Joint Space Control**



Manipulation (OM)



Using rospy: In a new terminal, type the command:

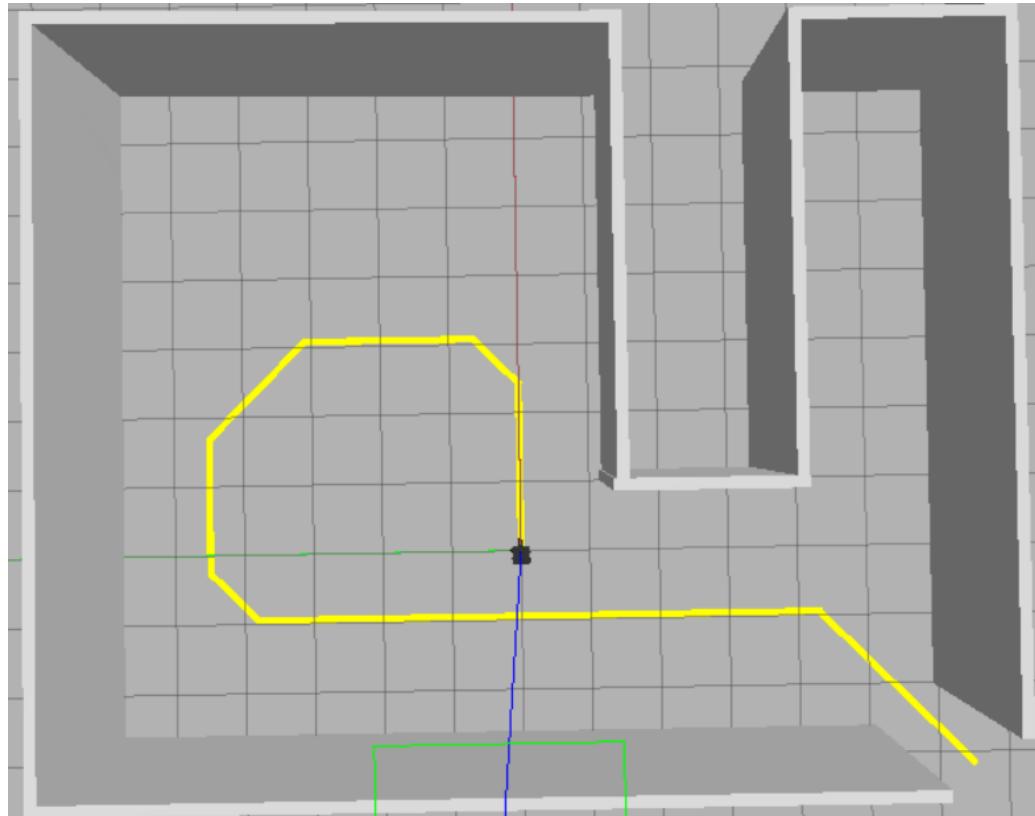
```
$ rosrun autonomous OM_DEMO.py
```

Similar to what was taught in the simulation module (M5), you can control the mounted OM using python codes:

- Change the values for the following variables; under the sub-functions *arm_joints1/2()*
joint_goal[X]
- Change the value for the following variable; under the sub-functions *gripper_open/close()*
joint_grip[0]



Machine Vision



- **TB3 with OM** robot example in Gazebo
- Aim of this exercise is for the robot to **track and follow the yellow line**
- We are going to do this in real life (i.e. physical TB3 with physical yellow lines)



Line Follower using MV Example



- Each group will be given **a bright yellow masking tape to mark your lines**
- You **do NOT** need the tunnel map for this exercise
- Find a space, mark your yellow lines using the given tape, and carry out the exercise (explained in next few slides)



Line Follower using MV Example



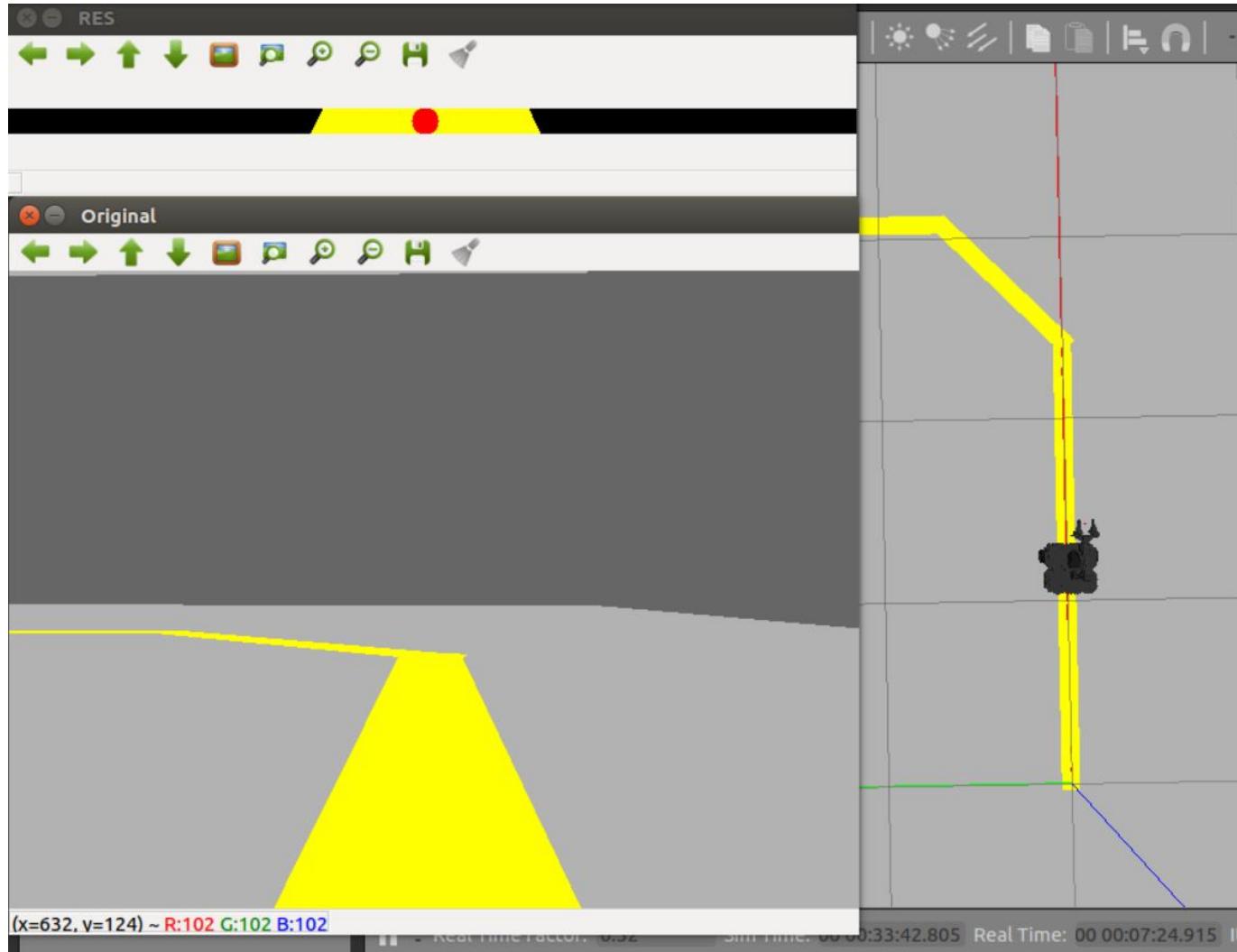
Procedures:

1. Place the TB3 at the start of the line with the camera facing the line
2. BringUp the TB3 (with OM) and activate the rpicamera (important step for this exercise!)
3. Compressed image should be transform to raw image (refer to README for command line to do this)
4. Run node example to utilize OpenCV to detect the yellow line and make the TB3 follow it

```
$ rosrun autonomous follow_line.py
```



Line Follower using MV Example

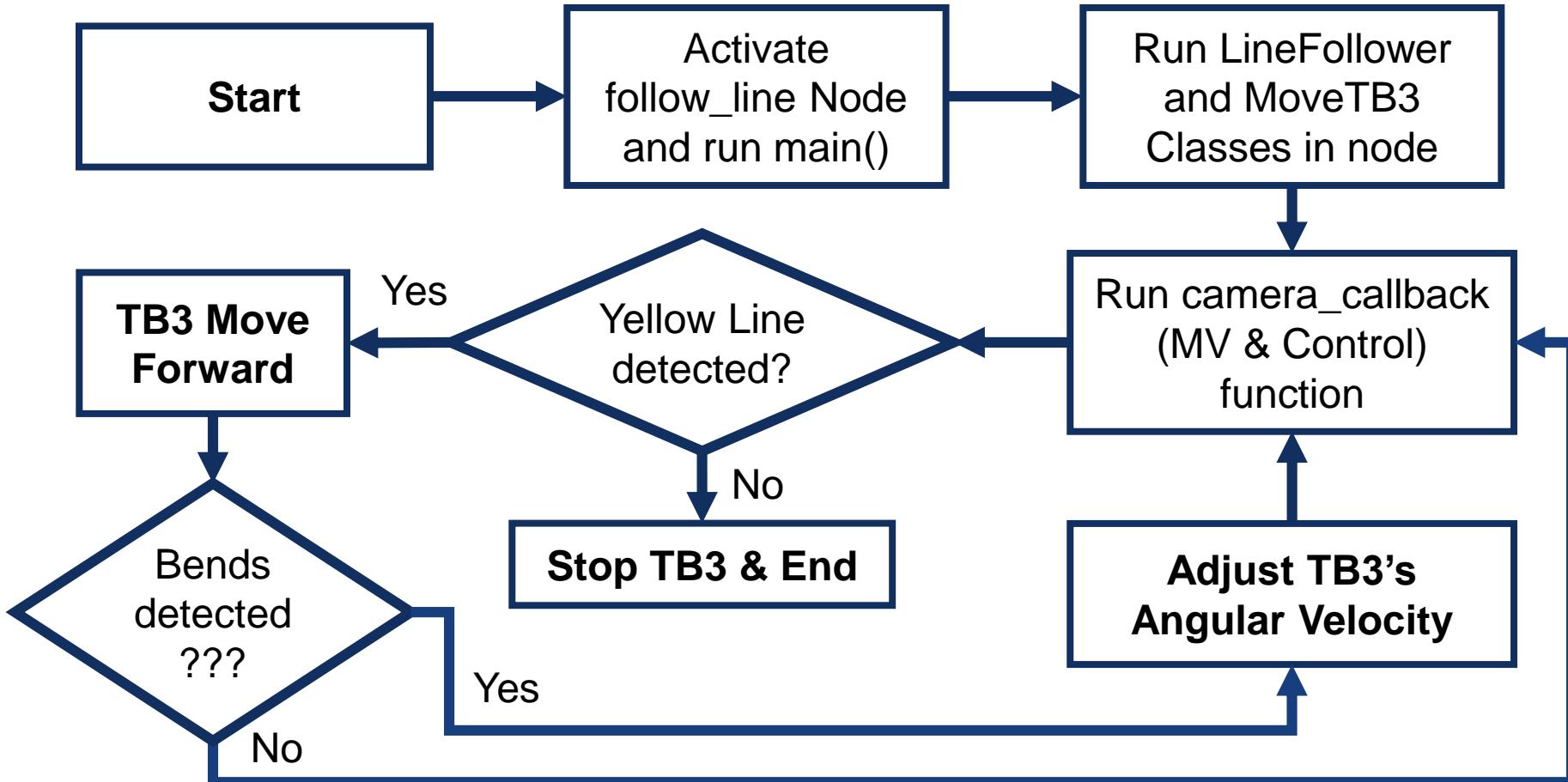




Line Follower using MV Example



Block Diagram Representation of Algorithm:





TURTLEBOT3 WAFFLE PI WITH ARM

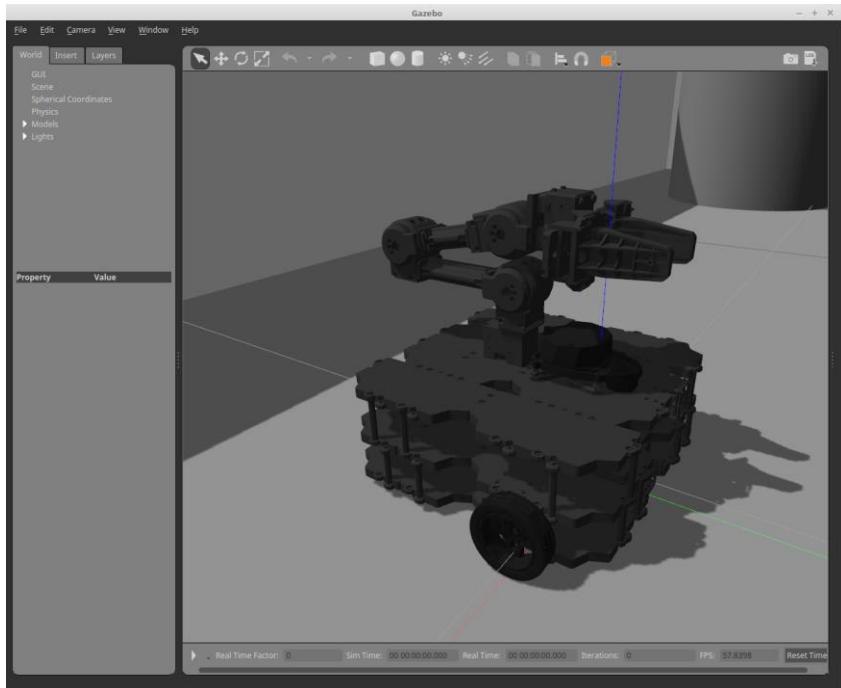
[VIRTUAL; TO PRACTICE]

Note that the physical TB3_with_OM system does not work well in ROS Noetic. Hence, your practice workshops for TB3_with_OM and Project 5 will be virtual-based instead.

The next few slides are there for your own information only; I will conduct a physical demo (based on ROS Kinetic) to showcase the workshops and Project 5 instead.



Run Gazebo



- Launch TurtleBot3 with arm in Gazebo
- \$ roslaunch turtlebot3_manipulation_gazebo turtlebot3_manipulation_gazebo.launch
- Click Play button



Run move_group Node



In a new terminal, type the command to use the Moveit feature:

```
$ roslaunch turtlebot3_manipulation_moveit_config  
move_group.launch
```

If the node is launched successfully, the following message will be printed in the terminal:

“You can start planning now!”

*****Important: you must run both Gazebo and move_group nodes first before you can control the OM on the TB3**

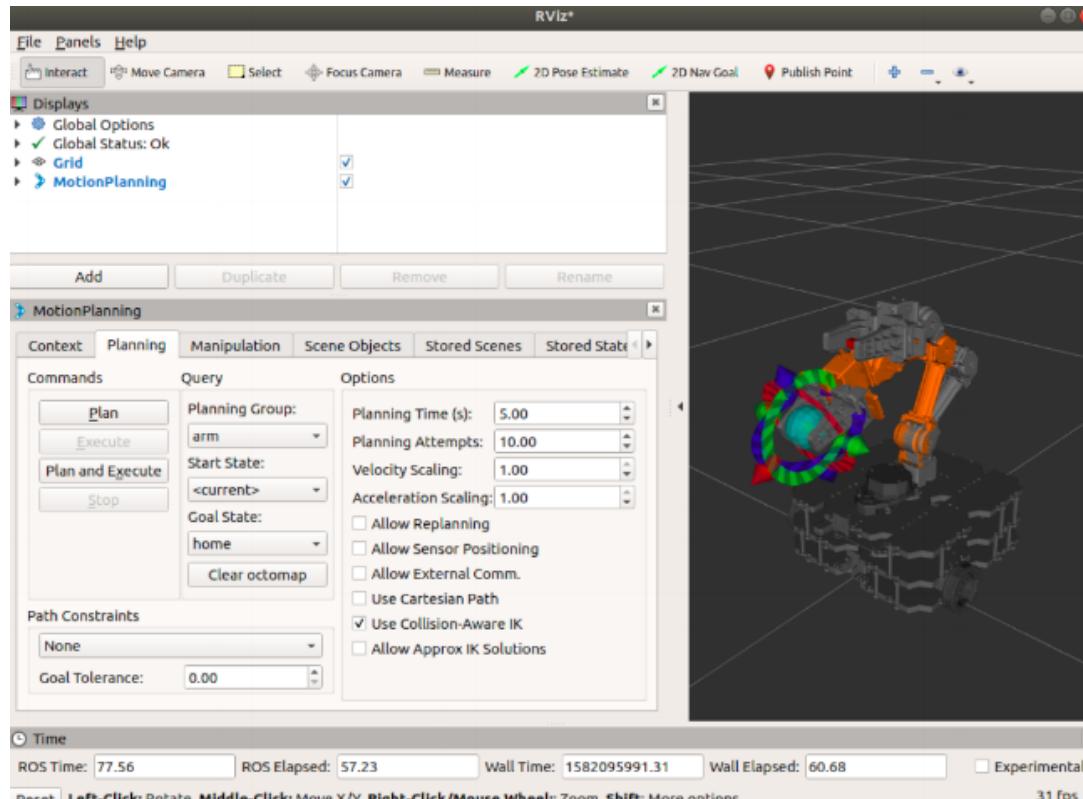


(a) Control via RViz



In a new terminal, type the command:

```
$ roslaunch turtlebot3_manipulation_moveit_config  
moveit_rviz.launch
```



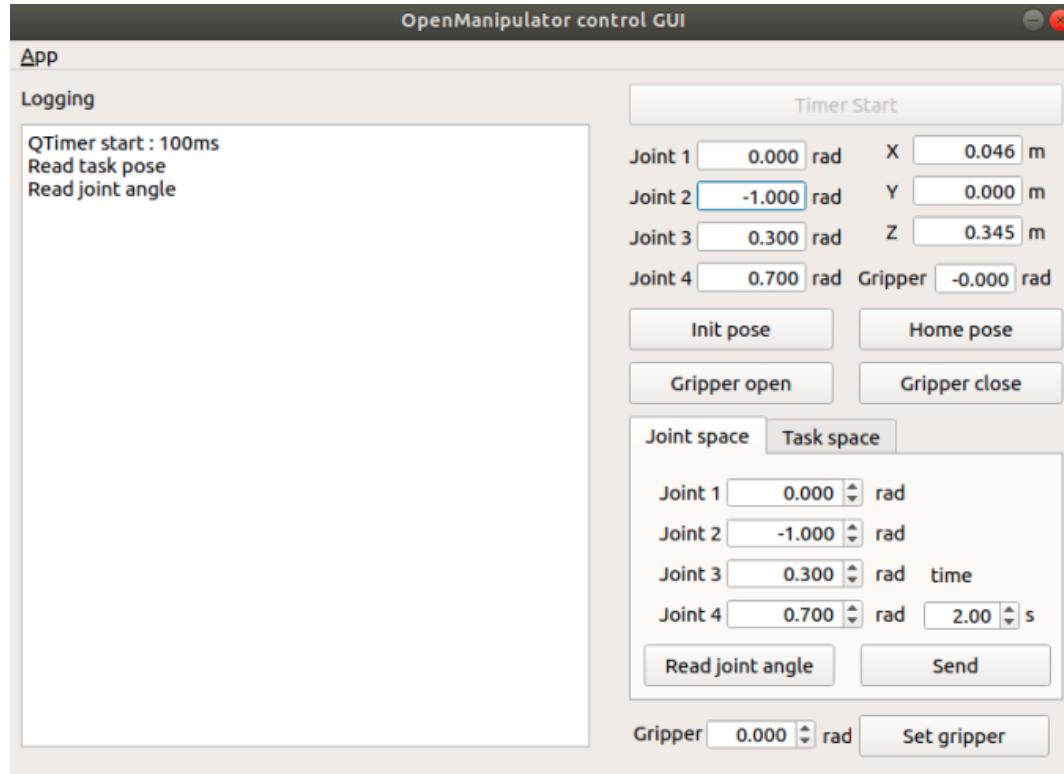
You can control the mounted OM using the Interactive Markers:

- Under MotionPlanning, click on the “Planning” Tab
- Move the Interactive Markers to shift the arm to your desired position
- Click “Plan and Execute” to move the arm

(b) Control via GUI

In a new terminal, type the command:

```
$ roslaunch turtlebot3_manipulation_gui  
turtlebot3_manipulation_gui.launch
```



- **Run Gazebo and the Move Group Node first**
- **Same way of using as the OM only**
- **Task Space Control and Joint Space Control**



(c) SLAM and Controlled Navigation



We will now use the following launch file:

TB3_with_OM.launch

This edited launch file allows you to use the TB3 world as your map in Gazebo for the TB3 with OM robot



(c) SLAM and Controlled Navigation



For Virtual SLAM (one TB3), 4 Steps; same as TB3 without arm but commands differ slightly (you may skip this process):

Step 1 - Launch the TB3 with arm in the *TB3 World* map in Gazebo

- \$ roslaunch autonomous TB3_with_OM.launch
- Click Play button

Step 2 - Launch RViz SLAM app in new terminal

- \$ roslaunch turtlebot3_manipulation_slam slam.launch



(c) SLAM and Controlled Navigation



Step 3 – Launch teleop node in a new terminal:

- `$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch`
- Use the keys to make the TB3 explore the area to capture data on the map

Step 4 - Save the Map once Completed in a new terminal:

- `$ rosrun map_server map_saver -f ~/map`



(c) SLAM and Controlled Navigation



For Virtual Controlled Navigation, 2 Steps:

Step 1 - Launch Gazebo in TB3 World

- \$ roslaunch autonomous TB3_with_OM.launch
- Click Play button

Step 2 - Execute Controlled Navigation in a new terminal

- \$ roslaunch turtlebot3_manipulation_navigation navigation.launch
- Click on “2D Pose Estimate” and set the correct initial position for the TB3 with arm
- Click on “2D NAV Goal” and set the target destination for the TB3 with arm

(c) SLAM and Controlled Navigation

RViz Navigation in the *TB3 World* map:



- **RECAP:** Due to the symmetrical effect within the environment, there will be unwanted cost maps after you have performed the initial pose
- Do remember to get rid of these unwanted cost maps before performing the navigation
- **Rotating and/or Moving the robot back and forth a bit first with teleop node**



(d) Rospy to control OM



For TB3_with_OM robot model, the system is created as such that **you cannot control the OM with messages (i.e. service, publish)**. Instead, you **have to control with Move Group nodes**. You can write a python file to control the OM with **Movelt interface**:

- Open terminal and type
`$ rosrun autonomous OM_moveit.py`
- The OM will move to a point determined by the joint angles. To alter these joint angles, open the python file, change the values for the following variables; under the sub-functions `arm_joints()`
`joint_goal[X]`
- To alter the gripper angle, change the value for the following variable (-0.01 for fully close and 0.01 for fully open); under the sub-functions `gripper_open/close()`
`joint_grip[0]`



(d) Rospy to control OM



OM_moveit.py (imports and Class):

```
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list

def all_close(goal, actual, tolerance):
    """
    Convenience method for testing if a list of values are within a tolerance
    of another list
    @param: goal       A list of floats, a Pose or a PoseStamped
    @param: actual     A list of floats, a Pose or a PoseStamped
    @param: tolerance  A float
    @returns: bool
    """
    all_equal = True
    if type(goal) is list:
        for index in range(len(goal)):
            if abs(actual[index] - goal[index]) > tolerance:
                return False

    elif type(goal) is geometry_msgs.msg.PoseStamped:
        return all_close(goal.pose, actual.pose, tolerance)

    elif type(goal) is geometry_msgs.msg.Pose:
        return all_close(pose_to_list(goal), pose_to_list(actual), tolerance)

    return True

class MoveGroupPythonInterfaceTutorial(object):
    """MoveGroupPythonInterfaceTutorial"""
    def __init__(self):
        super(MoveGroupPythonInterfaceTutorial, self).__init__()

    ## First initialize `moveit_commander`_ and a `rospy`_ node:
    moveit_commander.roscpp_initialize(sys.argv)
```

Note if you cant execute the python file, you have to give execution permissions to it by typing:

```
$ chmod +x name_of_the_file.py
```



(d) Rospy to control OM



OM_moveit.py (arm_joints1 sub function):

```
def arm_joints1(self):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a good
    # reason not to.
    group = self.group

    ## Planning to a Joint Goal
    # We can get the joint values from the group and adjust some of the values:
    joint_goal = group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = 0
    joint_goal[2] = 0
    joint_goal[3] = 0

    # The go command can be called with joint values, poses, or without any
    # parameters if you have already set the pose or joint target for the group
    group.go(joint_goal, wait=True)

    # Calling ``stop()`` ensures that there is no residual movement
    group.stop()

    ## END_SUB_TUTORIAL

    # For testing:
    # Note that since this section of code will not be included in the tutorials
    # we use the class variable rather than the copied state variable
    current_joints = self.group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)
```



(d) Rospy to control OM



OM_moveit.py (gripper_close sub function):

```
def gripper_close(self):

    grip = self.grip

    joint_grip = grip.get_current_joint_values()
    joint_grip[0] = -0.01

    grip.go(joint_grip, wait=True)

    grip.stop()

    current_grip = self.grip.get_current_joint_values()
    return all_close(joint_grip, current_grip, 0.01)
```



(d) Rospy to control OM



OM_moveit.py (main function):

```
def main():
    try:
        print "===== Setting up the moveit_commander ..."
        #raw_input()
        tutorial = MoveGroupPythonIntefaceTutorial()

        print "===== Execute movement using 1st joint state goal ..."
        #raw_input()
        tutorial.arm_joints1()
        time.sleep(7)

        print "===== Execute gripper open ..."
        #raw_input()
        tutorial.gripper_open()
        time.sleep(3)

        print "===== Execute movement using 2nd joint state goal ..."
        #raw_input()
        tutorial.arm_joints2()
        time.sleep(7)

        print "===== Execute gripper close ..."
        #raw_input()
        tutorial.gripper_close()
        time.sleep(3)

        print "===== Python tutorial demo complete!"
    except rospy.ROSInterruptException:
        return
    except KeyboardInterrupt:
        return
```



(d) Rospy to control OM



Try with a different python code:

```
$ rosrun autonomous OM_DEMO.py
```

Similar to what was taught previously, you can control the mounted OM using python codes:

- Change the values for the following variables; under the sub-functions *arm_joints1/2()*
joint_goal[X]
- Change the value for the following variable; under the sub-functions *gripper_open/close()*
joint_grip[0]



(e) Pick, Move & Place Example



Procedures:

1. Load TB3 with OM in TB3 World in Gazebo

```
$ roslaunch autonomous TB3_with_OM.launch
```

2. Run launch file example to **activate OM control (via move_group)**, to **activate the navigation node**, and to **activate the pick_move_place node**, which make OM “pick” first, after which move TB3 to a certain location, and lastly to make OM “place”

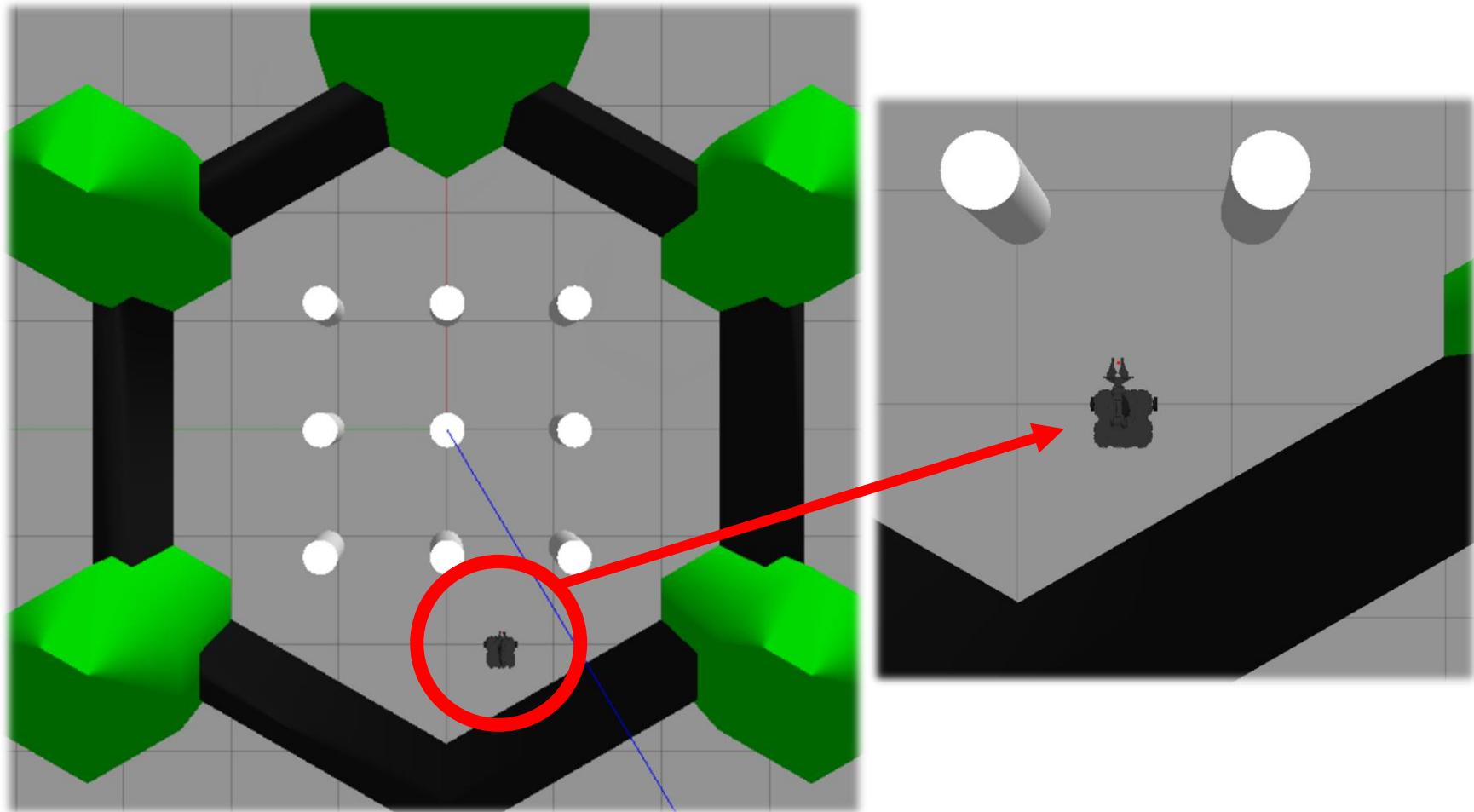
```
$ roslaunch autonomous pick_move_place.launch
```



(e) Pick, Move & Place Example



1. Load TB3 with OM in TB3 World in Gazebo:



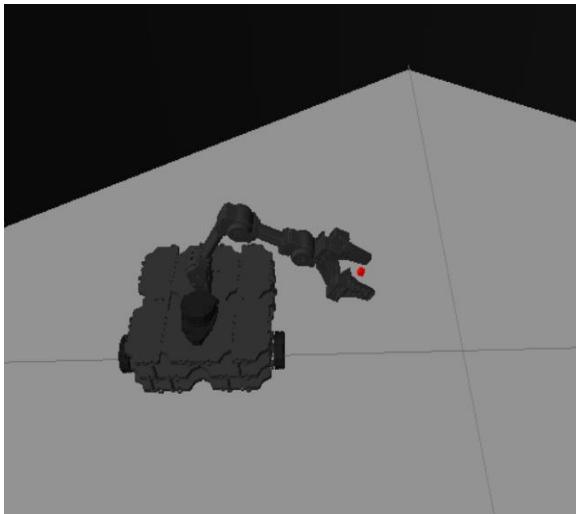


(e) Pick, Move & Place Example

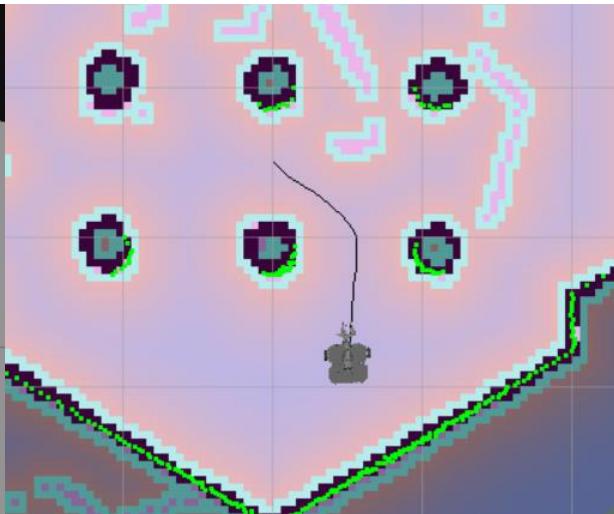


2. Run launch file example to “Pick”, Move and “Place”:

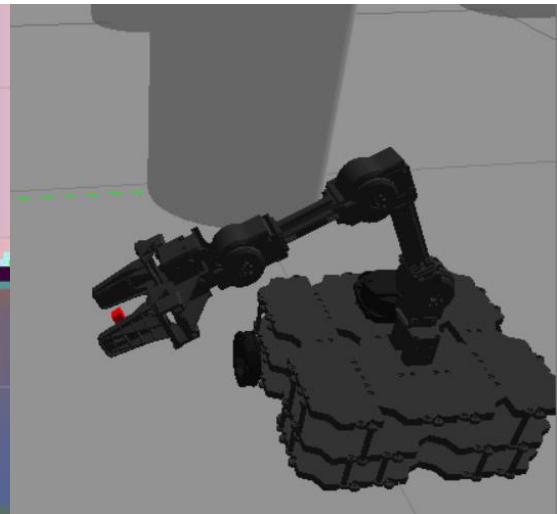
“Pick”



Move



“Place”

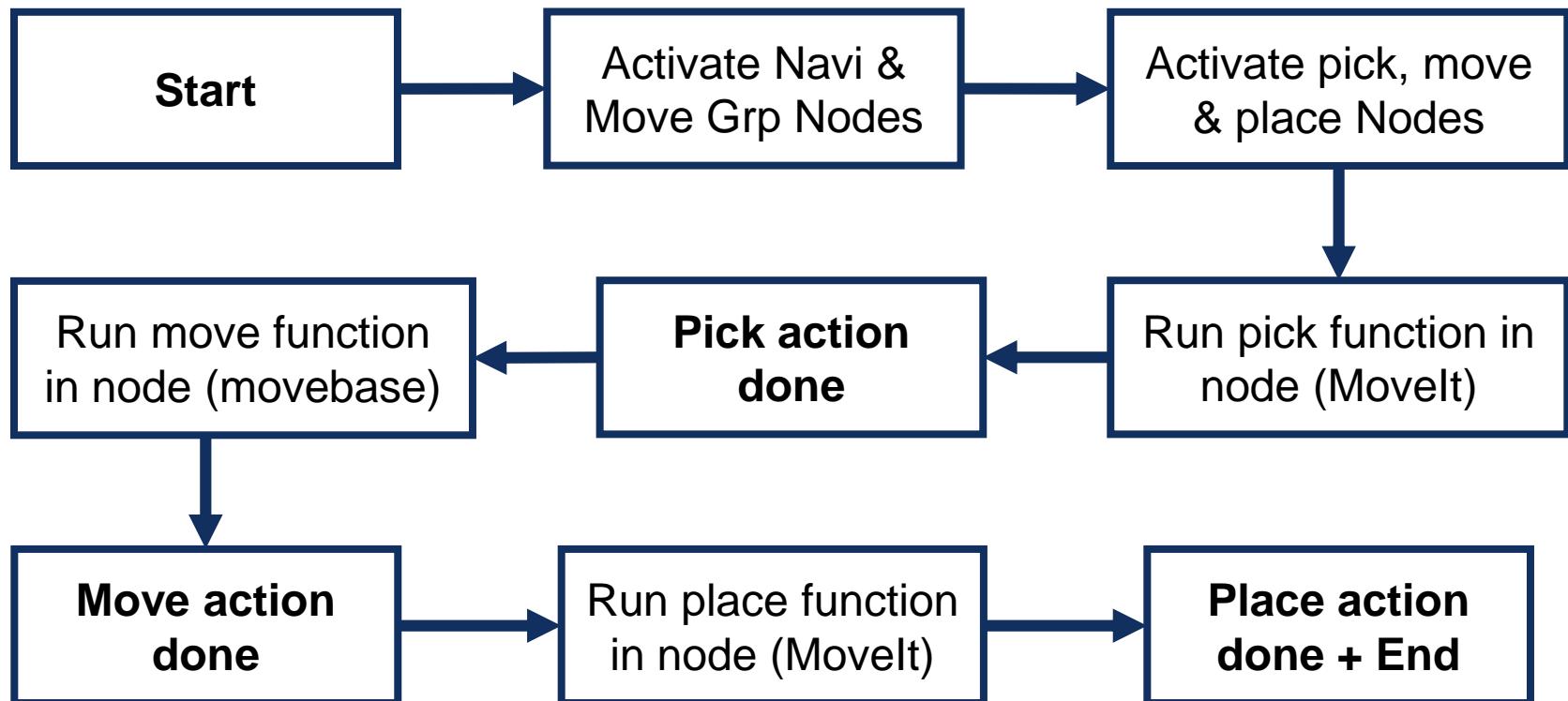




(e) Pick, Move & Place Example

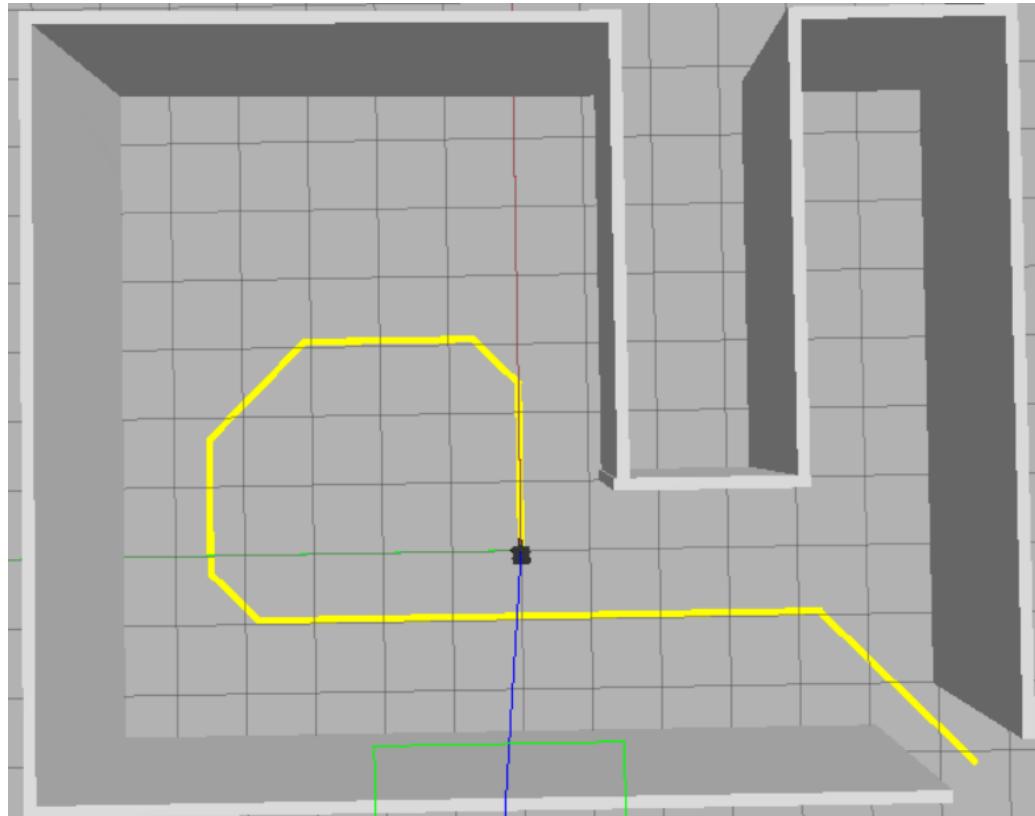


Block Diagram Representation of Algorithm:





(f) Machine Vision



- **TB3 with OM** robot example in Gazebo
- Aim of this exercise is for the robot to **track and follow the yellow line**
- We can also do this in real life (i.e. physical TB3 with physical yellow lines); to be shown in demo



(f) Line Follower using MV Example



Procedures:

1. Load TB3 with OM in line.world in Gazebo

[line2.launch, as compared to line.launch, is based on a slightly different map with shorter yellow lines; to save time]

```
$ roslaunch autonomous line2.launch
```

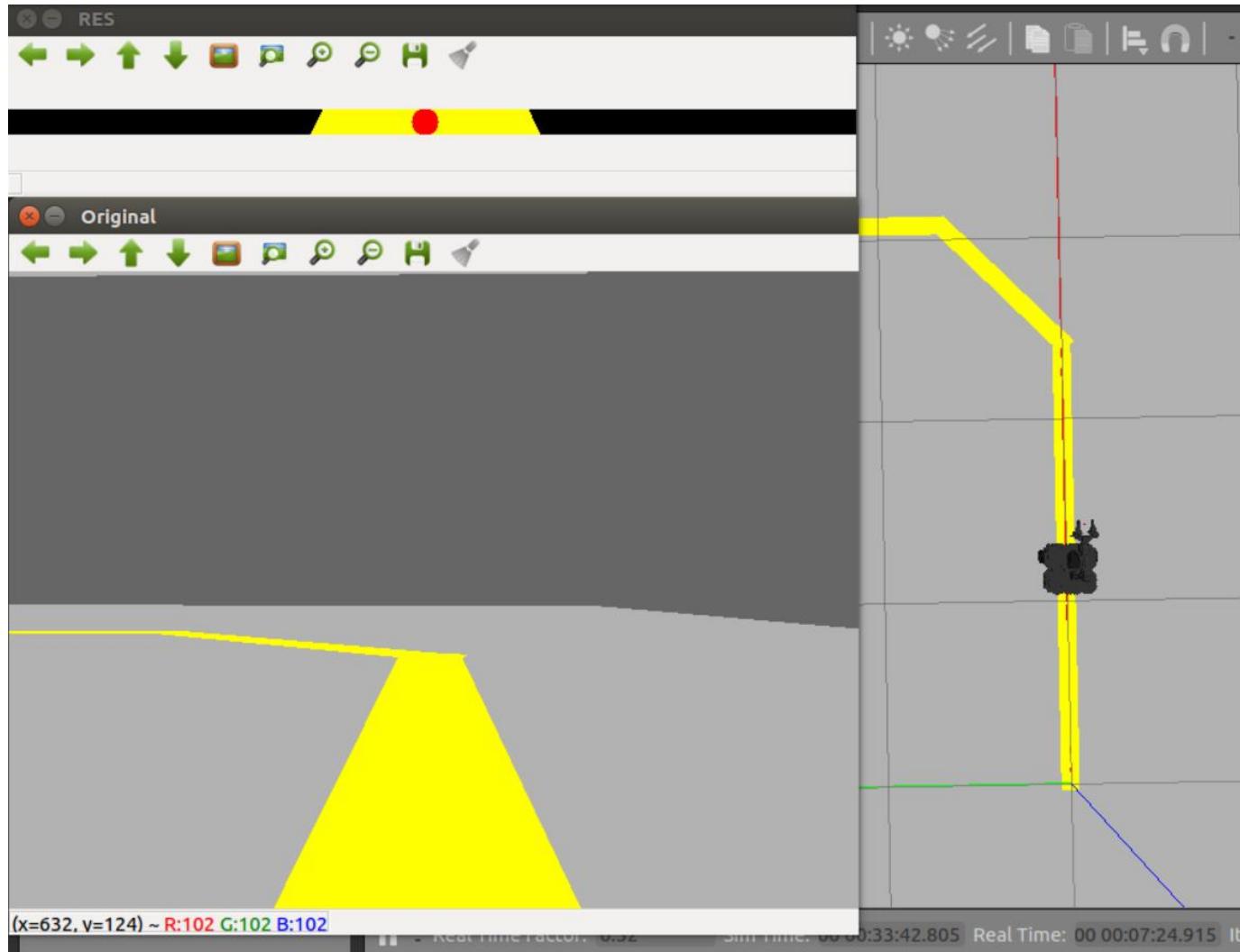
2. Run node example to utilize OpenCV to detect the yellow line and make the TB3 follow it

```
$ rosrun autonomous follow_line2.py
```

3. Examine the follow_line2.py file

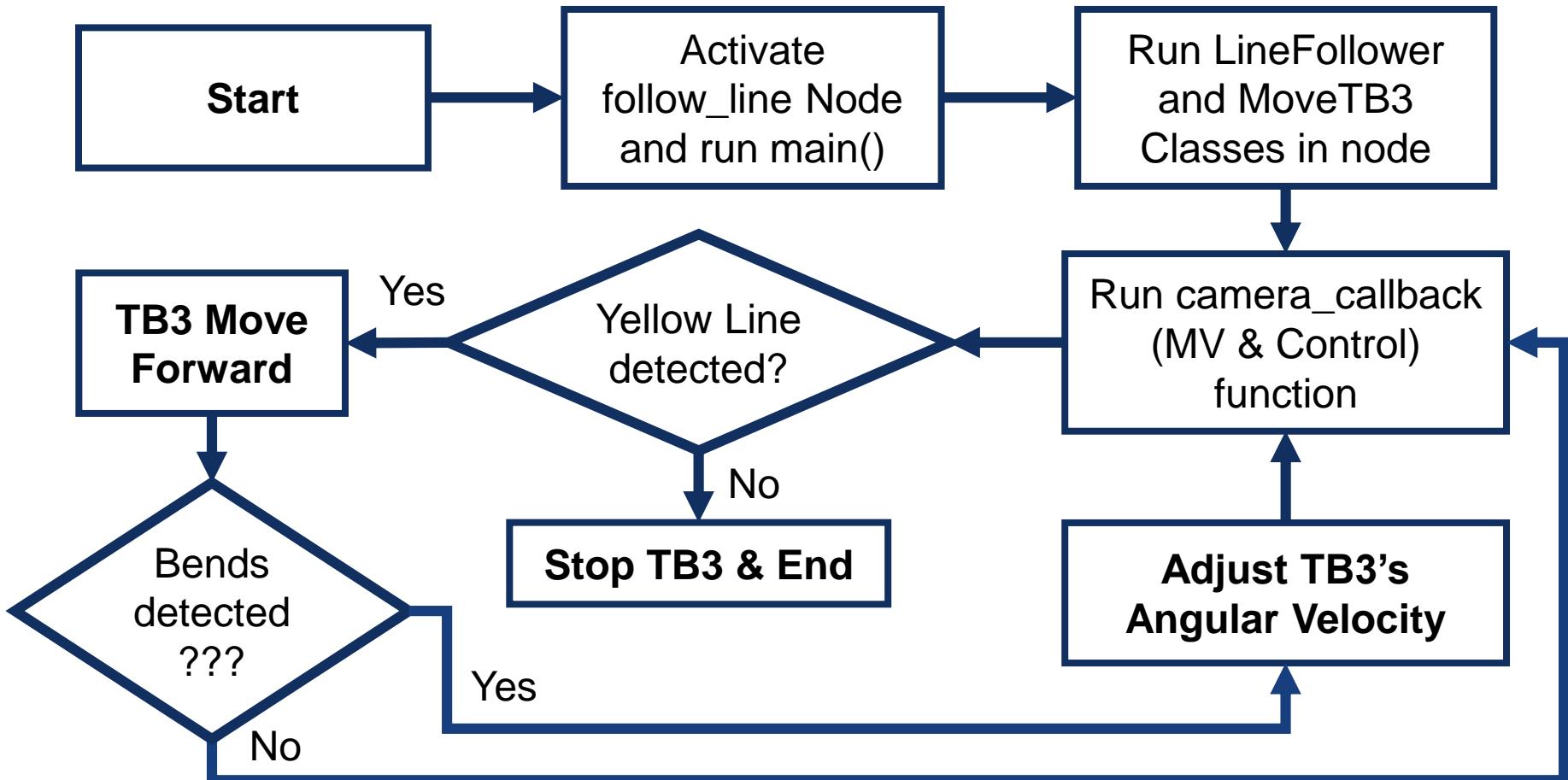


(f) Line Follower using MV Example



(f) Line Follower using MV Example

Block Diagram Representation of Algorithm:





GROUP PROJECT 5

APPLICATION OF MACHINE VISION & MANIPULATOR
OF TB3 WITHIN THE PHYSICAL WORLD



Group Project 5



The use of LIDAR has been demonstrated in the previous project. For this project, we will **explore the advantage of using a RGB camera** instead of the LIDAR for perception purposes to guide the TB3 from one location to another with the help of yellow lines.

We will also **explore the advantage of using a manipulator**. The use of manipulators in conjunction with vehicles is expected to increase the versatility of the autonomous system to operate functionalities of both arm and vehicle, which is useful in many applications.

You can **use the same layout that you have built from the previous exercise for this project**. The goal is to “pick” an item from the startpoint, move the TB3 base (together with the arm and “item”) to the endpoint **ONLY with the help of the RGB camera**, and “place” the item at the endpoint.



Group Project 5



In groups, you are to write a launch file that:

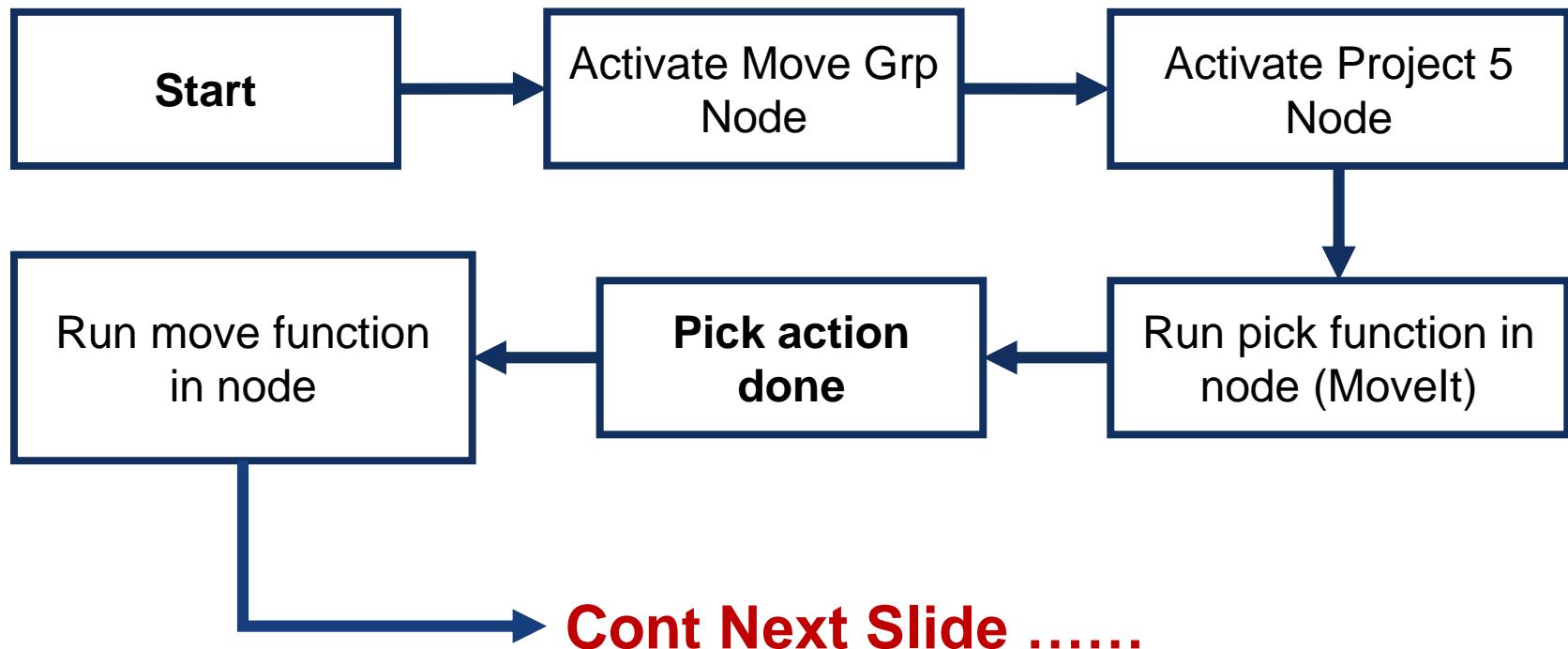
- 1. Activate the MANIPULATION node (i.e. move_group node) to get ready the OM for use**
- 2. Use the OM to “pick” the item from startpoint; assume this action remains the same as the “pick, move, place” example in M5**
- 3. Bring the TB3 with OM (together with the “item”) to the endpoint ONLY with the help of the RGB camera**
- 4. Use the OM to “place” the item at the endpoint; assume this action remains the same as the “pick, move, place” example in M5**



Group Project 5



Block Diagram Representation of Algorithm:



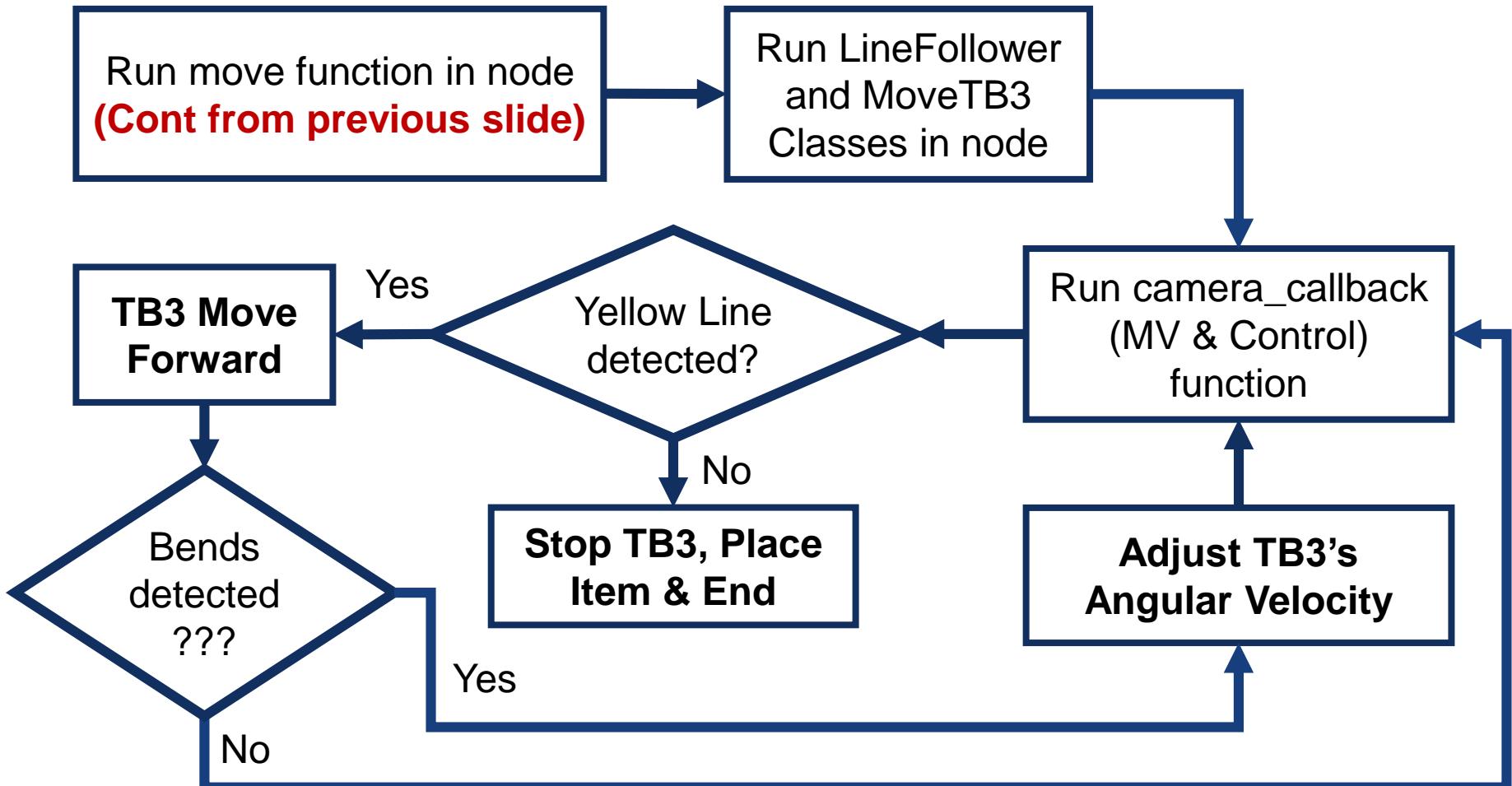
Cont Next Slide



Group Project 5



Block Diagram Representation of Algorithm:





Group Project 5



REMINDER [FYI below instructions are only applicable for physical robot; skip this]:

You have to **BRINGUP** your TB3 with OM (**2 BRINGUPs** to execute) first before you launch the launch file, Project5.launch (**refer to README for clearer and detailed instructions**)

Example:

1. BringUp the TB3 (with OM), activate the rpicamera (important step for this project!), transform image to RAW, BringUp the OM (with TB3)
2. Execute launch file to execute required tasks:

```
$ roslaunch autonomous Project5.launch
```

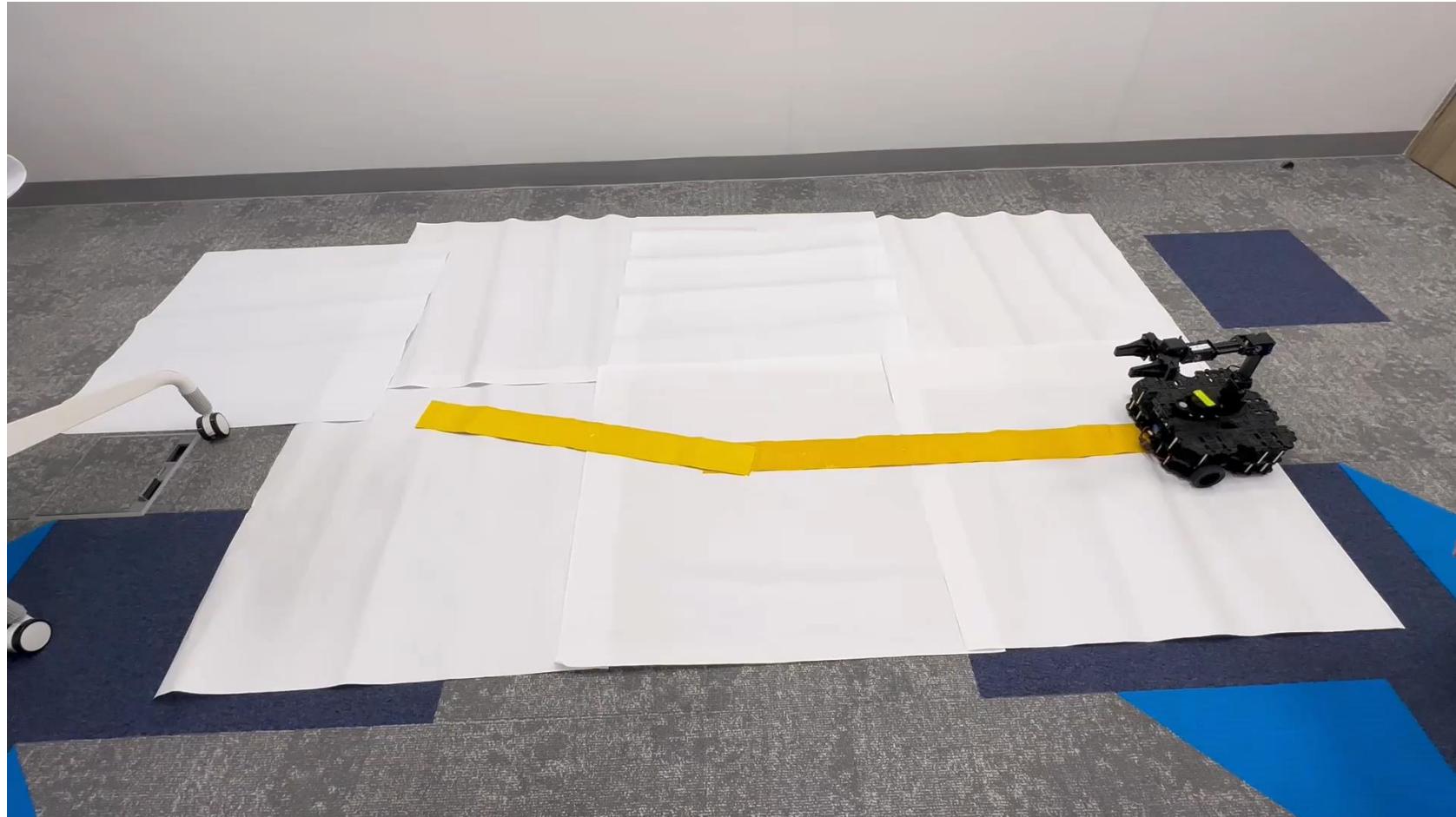
Meaning you are only required to type 1 command line to perform the given tasks



Group Project 5



DEMO VIDEO on Physical Robot (ROS Kinetic):





Group Project 5



Reflect & answer the questions below:

1. What are the advantages of utilizing RGB cameras in autonomous systems as compared to the LIDAR? Compare in terms of your observations from P4 and P5
2. What are the advantages of utilizing manipulators in autonomous vehicles or the like?



Group Project 5



3 Items to Submit (can zip all in one file):

1. **Project5.launch**
2. **Other supporting python file(s)**
3. **Reflection on P5 (word or PDF accepted)**



Please upload all your works
[Project 4 & 5 zipped together] in
CANVAS → Assignments →
ARV Workshop 4 Submission

(Each representative per group to submit)



End of Module 6



**THANK YOU
for your kind
attention!**



Main References



- <https://mechathon.com/components-parts-of-automobile/>
- <http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>
- http://emanual.robotis.com/docs/en/platform/open_manipulator_x/overview/
- ROS Robot Programming: From the basic concept to practical programming and robot application by Pyo, et al.