MODULE 3:
TECHNICAL FUNDAMENTALS – AUTONOMOUS VEHICLES & ROBOTICS TECHNOLOGY

Nicholas Ho

Institute of System Science, NUS

Contents

1. Understanding the general architecture of Autonomous Systems
2. Challenges faced while designing or developing Autonomous Systems
3. Other aspects of Autonomous Systems Design
4. Understanding the concept of technologies and system architectures that are involved in Autonomous Vehicles and Robots
5. Workshop: Develop basic autonomous robot arm

CHAPTER 1:
UNDERSTANDING THE GENERAL ARCHITECTURE OF AUTONOMOUS SYSTEMS

Autonomous Systems as The Big Picture

Source: https://www.youtube.com/watch?v=TI7pQ5zSEJs

Concept of Autonomy

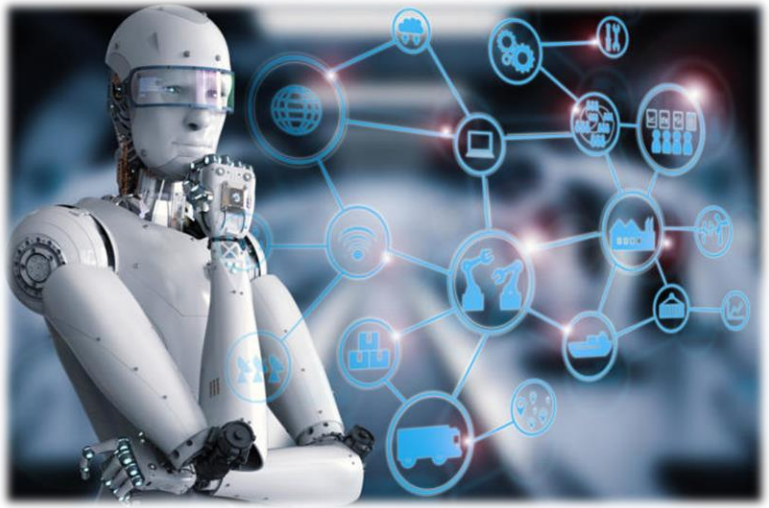
- **Autonomy = Ability to make one's own decisions**
- **Key to the IoT vision** promising increasing integration of smart systems to achieve goals such as optimal resource management and enhanced quality of life, with **minimal human intervention**
- **Very important questions for autonomous systems engineering:**
  1. Is there a **general reference model** that could provide a basis for evaluating system autonomy?
  2. What are the **technical solutions** for enhancing a system's autonomy?
  3. For each enhancement, is it possible to **estimate the implied technical difficulties and risks**?

Concept of Autonomy

ISO 29990:2010

Discussion Question:

Are autonomous systems ALL about AI and learning techniques?

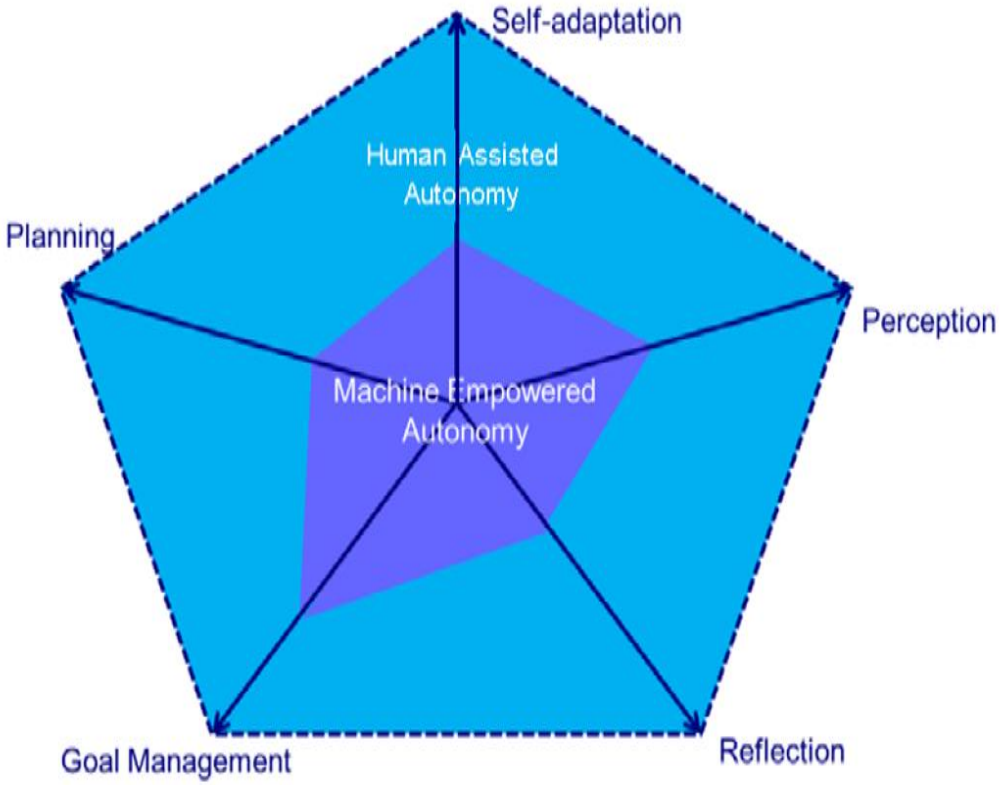


Concept of Autonomy

Any differences among the following?

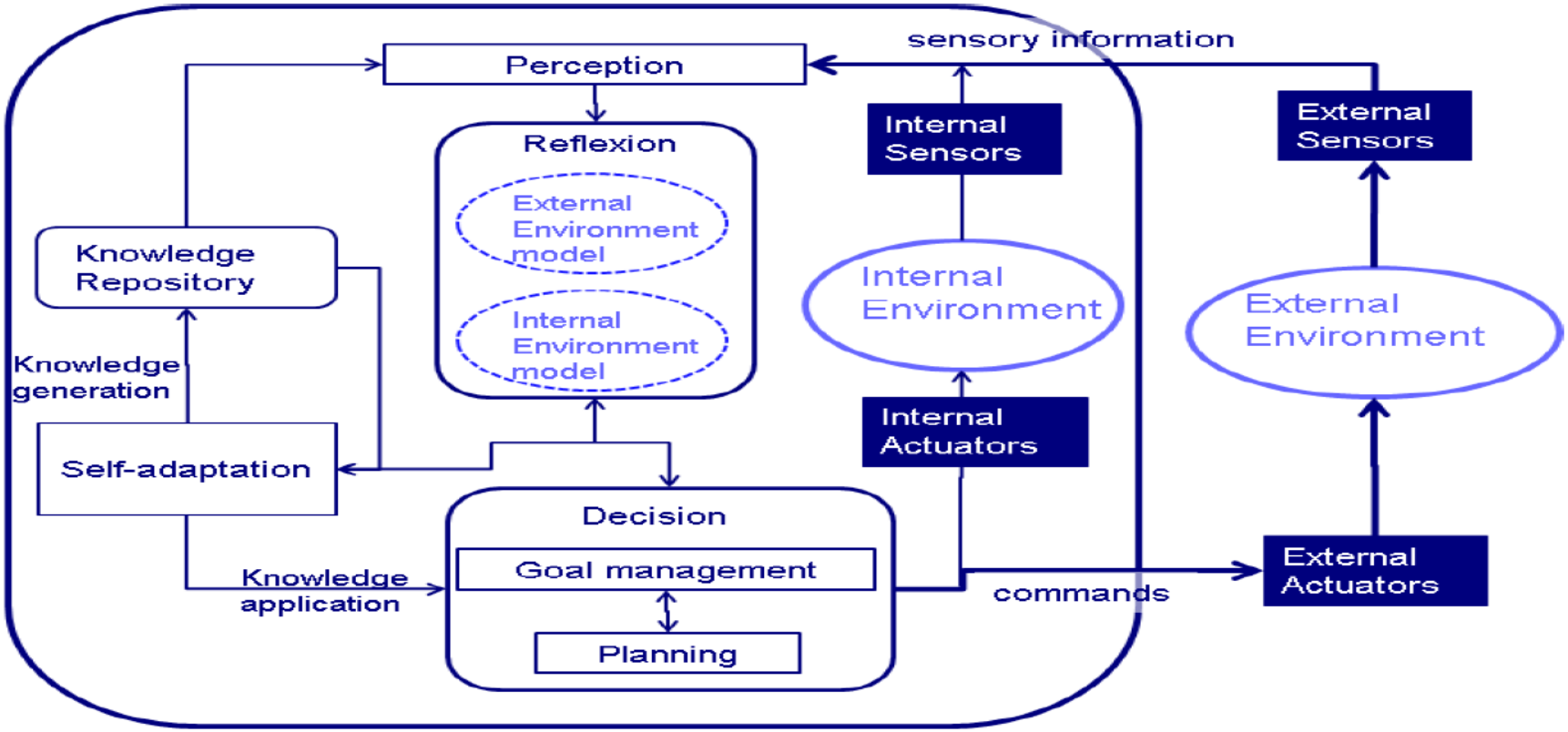
Hint: 3 Main Differences

Main Aspects of Autonomy

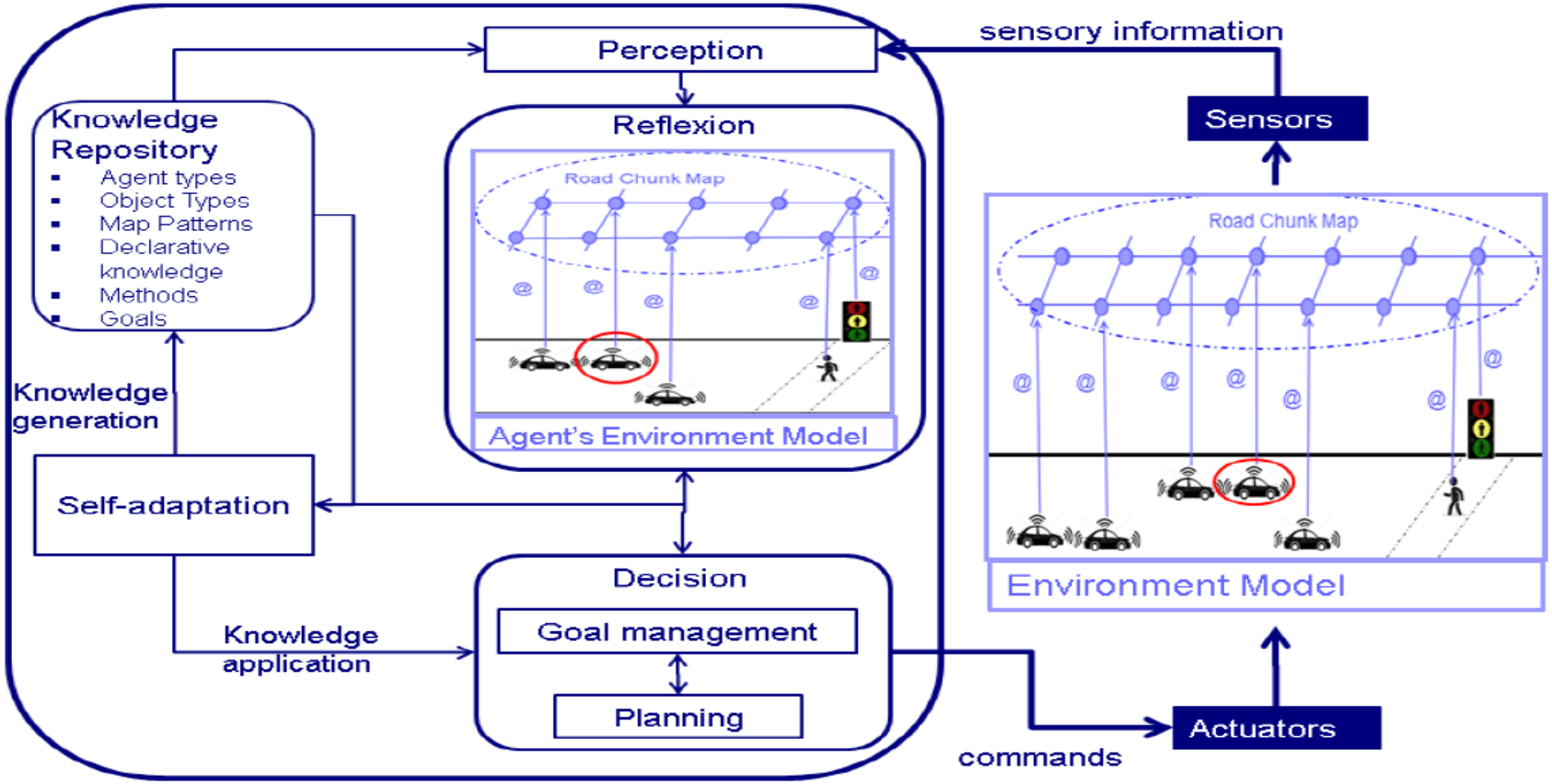


1. **Perception** e.g. interpretation of stimuli, removing ambiguity/vagueness from complex input data and determining relevant information
2. **Reflection** e.g. building/updating a faithful environment run-time model
3. **Goal management** e.g. choosing among possible goals the most appropriate ones for a given configuration of the environment model
4. **Planning** to achieve chosen goals
5. **Self-adaptation** e.g. the ability to adjust behavior through learning and reasoning and to change dynamically the goal management and planning processes

General Architecture of Autonomous Systems



General Architecture of Autonomous Systems



Applied to Autonomous Vehicles Case

General Architecture of Autonomous Systems

1. The Knowledge Repository

- **Contains various types of knowledge** used by the other modules for:
  a) the interpretation of sensory information;
  b) building the environment model of the agent;
  c) goal management and subsequent goal planning
- **Some is developed at design time (static), and some is produced and stored at run time (dynamic)**

General Architecture of Autonomous Systems

1. The Knowledge Repository (Cont)

- **Examples of design time knowledge:**
  a) list of all the relevant types of agents and objects and their corresponding behaviors
  b) list of predefined maps and coordination patterns
  c) list of the goals
  d) monitoring and learning techniques
- **Examples of run time knowledge:**
  a) Properties of the agent model
  b) Knowledge produced by monitors of the agent's behavior
  c) Knowledge produced by application of learning techniques

General Architecture of Autonomous Systems

2. The Perception Module

- **Extracts relevant information from the various stimuli provided by sensors**; extracted information is linked to the knowledge repository
- Makes use of learning techniques or of analysis and recognition processes
- **It concerns:**
  a) the type and identity of each sensed agent or object
  b) the state of the identified components
  c) the type of the external environment
- **Example:** Perception module of a self-driving car provides the types of the components in the vicinity with their associated attributes

General Architecture of Autonomous Systems

3. The Reflection Module

- **Uses information provided by the Perception module to build/update the environment models** (i.e. external and internal)
- **Very important for agents with dynamically changing environment** (e.g. self-driving cars)
- Extensively uses design time knowledge of the Repository to build a complete behavioral model of its perceived environment
- **Performance of this module is critical for mobile agents subject to real-time constraints**; How fast the agent's environment model can track changes of the real environment?

General Architecture of Autonomous Systems

4. The Decision Module

- **Two cooperating submodules:**
  a) Goal Manager that handles the actual agent's goals
  b) Planner that generates plans that implement particular goals
- **Assigns higher priority to critical goals according to their importance**
- **Goal management = Solving an optimization problem**
  - ❖ translates goals into utility function policies: a goal is characterized as the desired set of feasible states for which the objective function is optimized subject to a set of constraints
- **For a selected goal, the Planner computes from the environment model a corresponding plan** (involves commands for interaction with other agents or reconfiguration of their environment)

General Architecture of Autonomous Systems

5. The Self-adaption Module

- **Supervises and coordinates all the other modules**
- **Continuously reassesses the coherency of the exchanged information, creates new knowledge and provides directives to the Goal manager**
- **Combines reasoning and run-time analysis techniques; examples:**
  a) monitoring or analysis techniques to the environment model to detect critical situations
  b) learning techniques to estimate parameters or detect abnormal situations
- **Directives to the Goal Manager include:**
  a) Change of parameters
  b) Change of the set of the managed goals

CHAPTER 2:
CHALLENGES FACED WHILE DESIGNING OR DEVELOPING AUTONOMOUS SYSTEMS

Possible Challenges

1. **Design Complexity**
2. **Autonomy Complexity**
3. **Implementation Complexity**
4. **Chaotic environments (limitless scenarios)**
5. **Challenging conditions (e.g. rain, snow, low sunlight, mist)**
6. **Miscommunications between the system and human user**

1. DESIGN COMPLEXITY

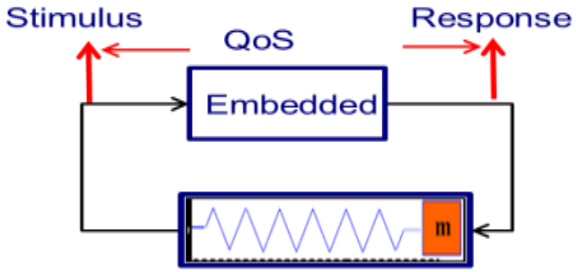
Design Complexity of Autonomous Systems

Reactive Complexity

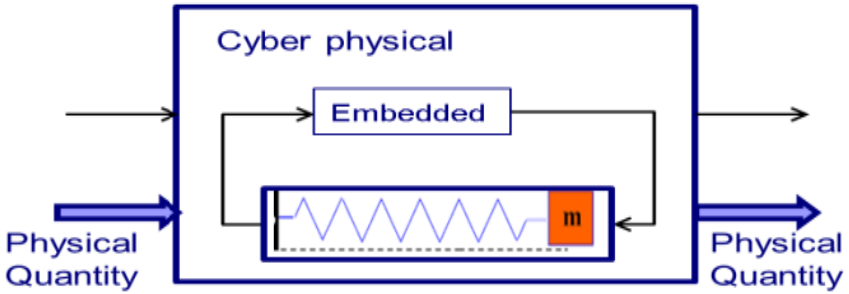
- Characterizes the **complexity of the interaction between an agent and its environment**

Transformational agent e.g.
Intelligent Personal Assistant

Streaming Agent
e.g. Encoder, Signal processor



Embedded Agent
e.g. Flight controller



Cyber physical agent
e.g. Self-driving car

Architecture Complexity

Characterizes the **complexity of coordination between agent(s) and object(s)**; type of architectures with increasing complexity:

a) **Static:** A given number of agents and objects, with fixed coordinates

b) **Parametric:** Arbitrary initially known numbers of "pluggable" components for fixed coordination patterns

c) **Dynamic:** Parametric architectures with dynamic creation/deletion of agents or objects

d) **Mobile:** Dynamic architectures where also the coordinates of objects and agents can change dynamically (e.g. swarm robotic system). May involve dynamic change of maps when mobile agents explore a space and progressively build a model of their environment.

e) **Self-organizing:** Mobile architectures with many dynamically changing motifs (e.g. for robocars, soccer playing robots). Able to adapt coordination rules to changing system dynamics.

P.S. Objects are physical dynamic systems without computation capability

2. AUTONOMY COMPLEXITY

Autonomy Complexity

1. **Complexity of perception:** **Difficulty to interpret stimuli provided by the environment** and to **timely generate inputs for the agent environment model**. Caused by stimuli ambiguity (admitting different interpretations), vagueness (fuzzy or noisy stimuli), high volume of stimuli data
2. **Lack of observability/controllability:** **Partial knowledge of the agent's environment**. **Affects the ability to build plans and act on the environment**
3. **Complexity of goal management:** Complexity of the process of **choosing the most compatible set of goals**. Utilization of prioritization and optimization models
4. **Complexity of planning:** **Directly depends on the type of goals and the complexity of the environment**; goals can be simple or complicated
5. **Complexity of adaptation:** Directly related to **uncertainty about the environment**

3. IMPLEMENTATION COMPLEXITY

Implementation Complexity

1. **Choosing the Most Suitable Architecture Type to Implement**
   - **Centralized, Decentralized, Distributed**
2. **Accuracy of Dynamic Model of the System**
   - Challenging to design a **reliable controller**
   - Hence, challenging to properly control the system
3. **Issues of having Multiple Agents**
   - **Coordination issues**
   - **Conflicting objectives**

4. CHAOTIC ENVIRONMENTS

Chaotic Environments

To be continued in HRSE course ……

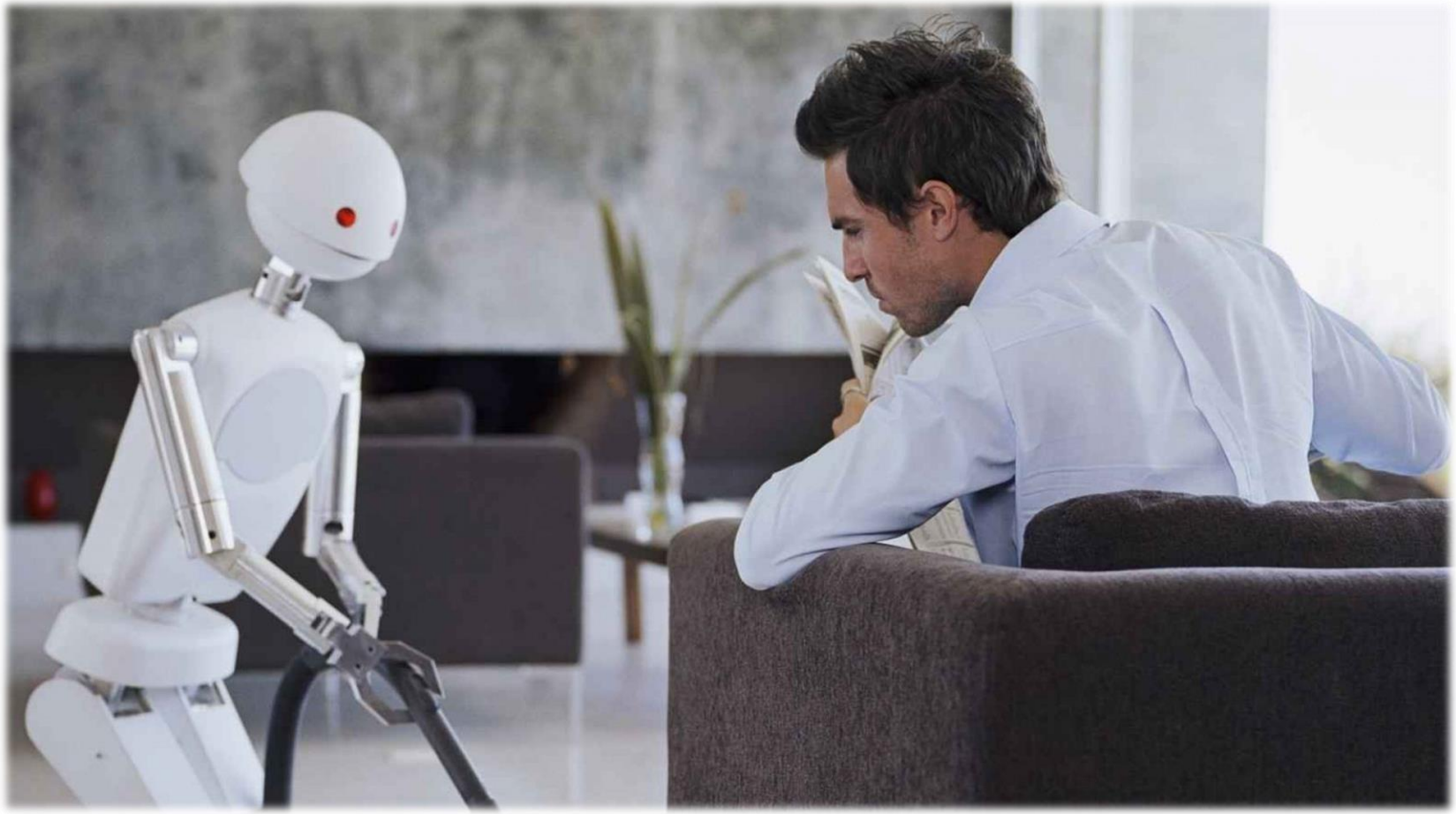
5. CHALLENGING CONDITIONS

Challenging Conditions

Conditions that impaired view of system; possible risk of safety compromise

6. MISCOMMUNICATIONS BETWEEN THE SYSTEM AND HUMAN USER

Miscommunications



E.g. Clearing rubbish in your house

CHAPTER 3:
OTHER ASPECTS OF AUTONOMOUS SYSTEMS DESIGN

Funny Self-Driving Car Commercial

Source: https://www.youtube.com/watch?v=iRIn4-5TmUM

Other Aspects of AS Design

1. Systems Safety
2. Cyber Security
3. Reliability
4. Responsibility & Accountability

1. SYSTEMS SAFETY

System Safety

- **Most basic requirement** of autonomous systems
- **Questions to ask:**
  1. How should the system be tested?
  2. What guidelines should be fulfilled to ensure that it is safe to use? E.g. standards
  3. For AV case, should a self-driving car require a driver license?
- **Economic considerations might compromise the safety of the system** (e.g. cheap equipment, negligence to maintain software)

System Safety

Possible Solutions:

1. Failure Mode, Effects & Criticality Analysis (FMECA)

- Used to chart the probability of failure modes against the severity of their consequences; performed at either the functional or piece-part level
- Result highlights failure modes with relatively high probability and severity of consequences

2. Fault Tree Analysis

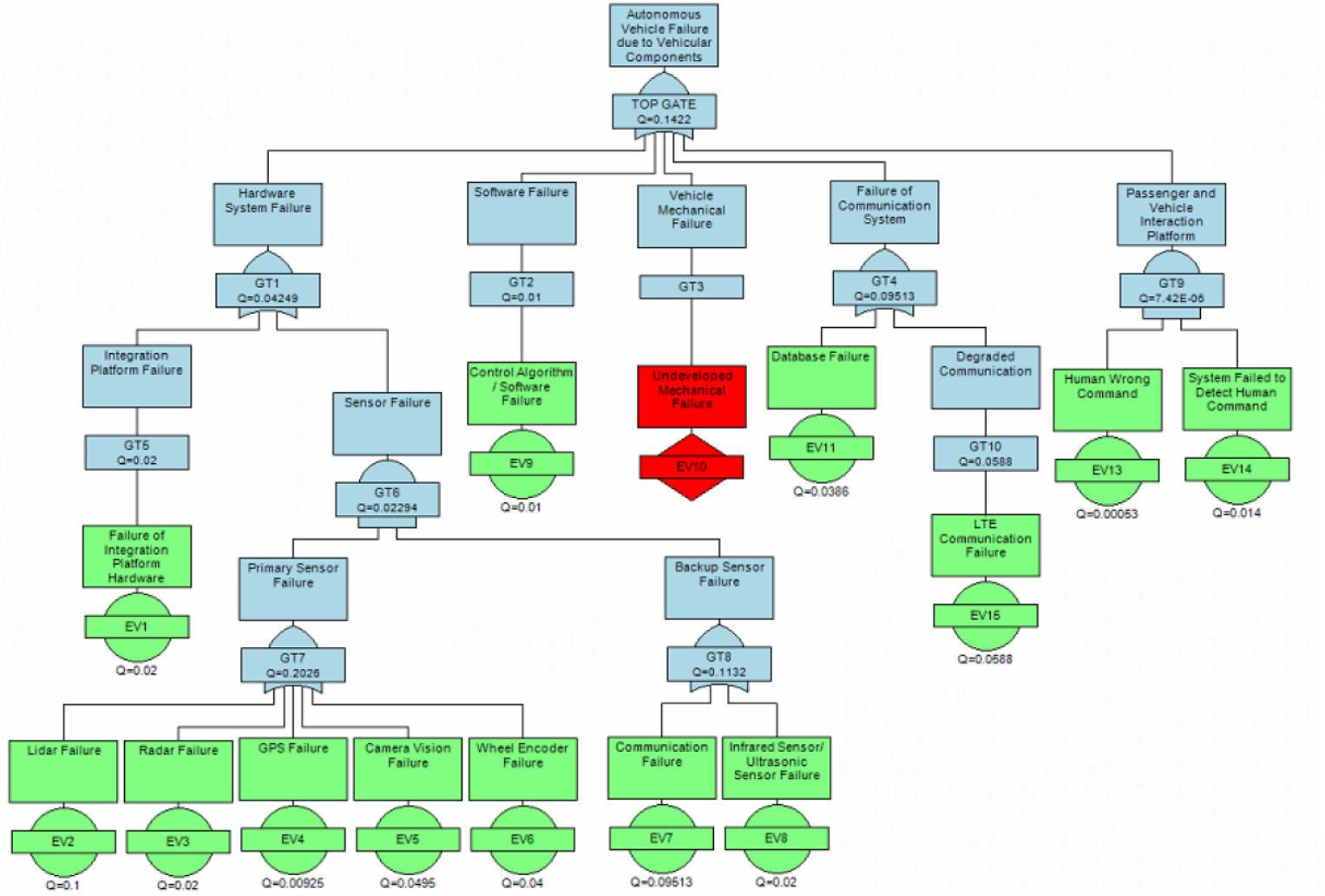
- Utilized to perform risk estimation because of its capability to provide the shortest path to reach the top-level failure from a single component failure

3. Dependent Failure Analysis (DFA)

- Aims at identifying failures that may hamper the required independence from interference between given elements (hardware/ software/ firmware) which can compromise system safety

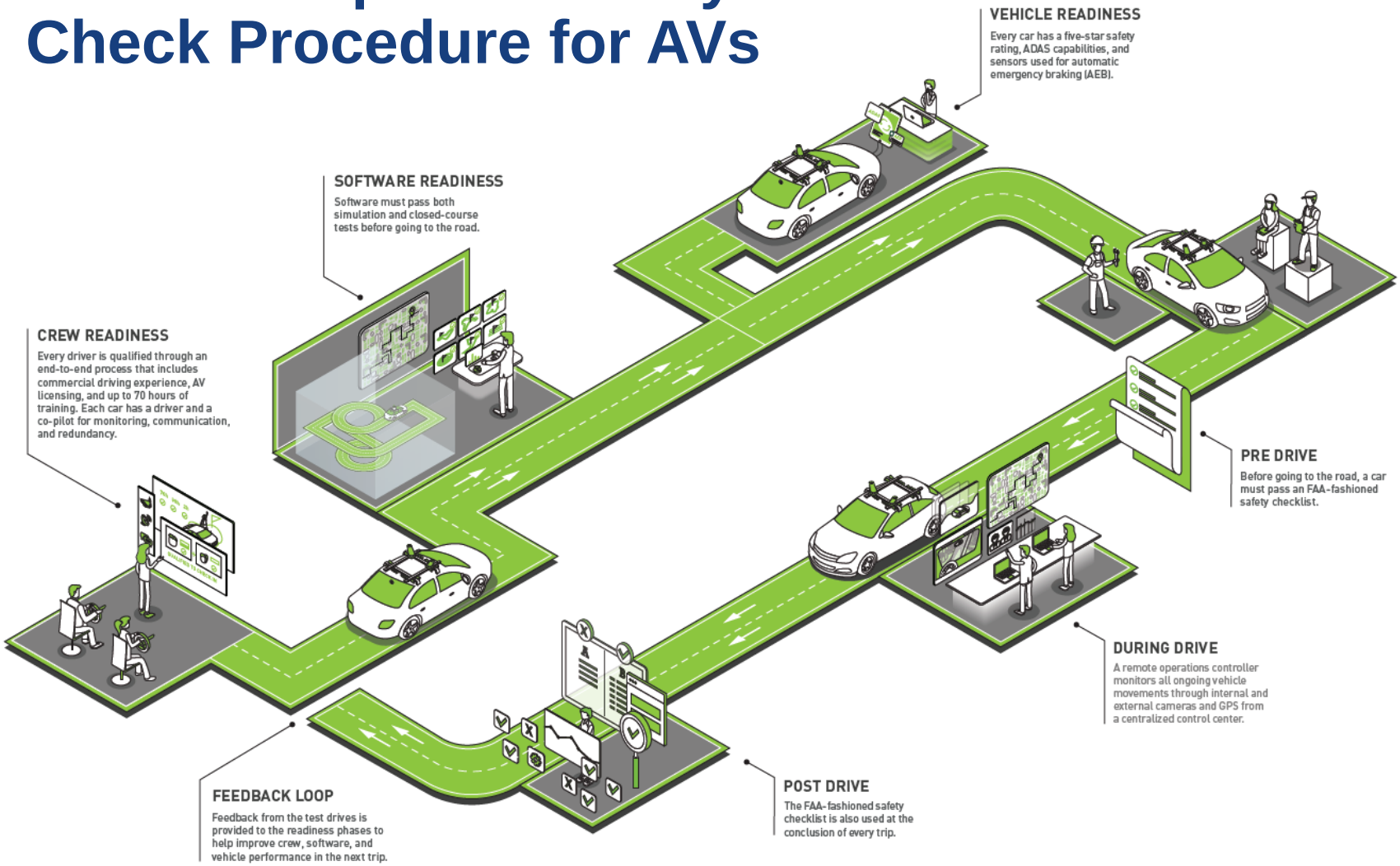
ISO 29990:2010

Fault Tree Analysis Example for AV:



System Safety:
NVIDIA Proposed Safety Check Procedure for AVs

ISO 29990:2010



2. CYBER SECURITY

Cyber Security

WIRED

Hackers Remotely Kill a Jeep on the Highway—With Me in It

ANDY GREENBERG SECURITY 07.21.15 6:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

ANDY GREENBERG SECURITY 09.10.15 7:00 AM

GM TOOK 5 YEARS TO FIX A FULL-TAKEOVER HACK IN MILLIONS OF ONSTAR CARS

5 » Tesla Patches Vulnerabilities that Allowed Remote Takeover of...

Technology IPhone Android

BMW ConnectedDrive ha... cars exposed to remote u...

Tesla Patches Vulnerabilities that ...over of

International Business

News World Business Fintech Politics Technology Science Sport Entertainm

Hackers can control ... heating and access d...

Weakness in communication between ele... opportunities for hackers

Technology CyberSecurity

Hackers disable Corvette brakes dongle meant to lower insurance

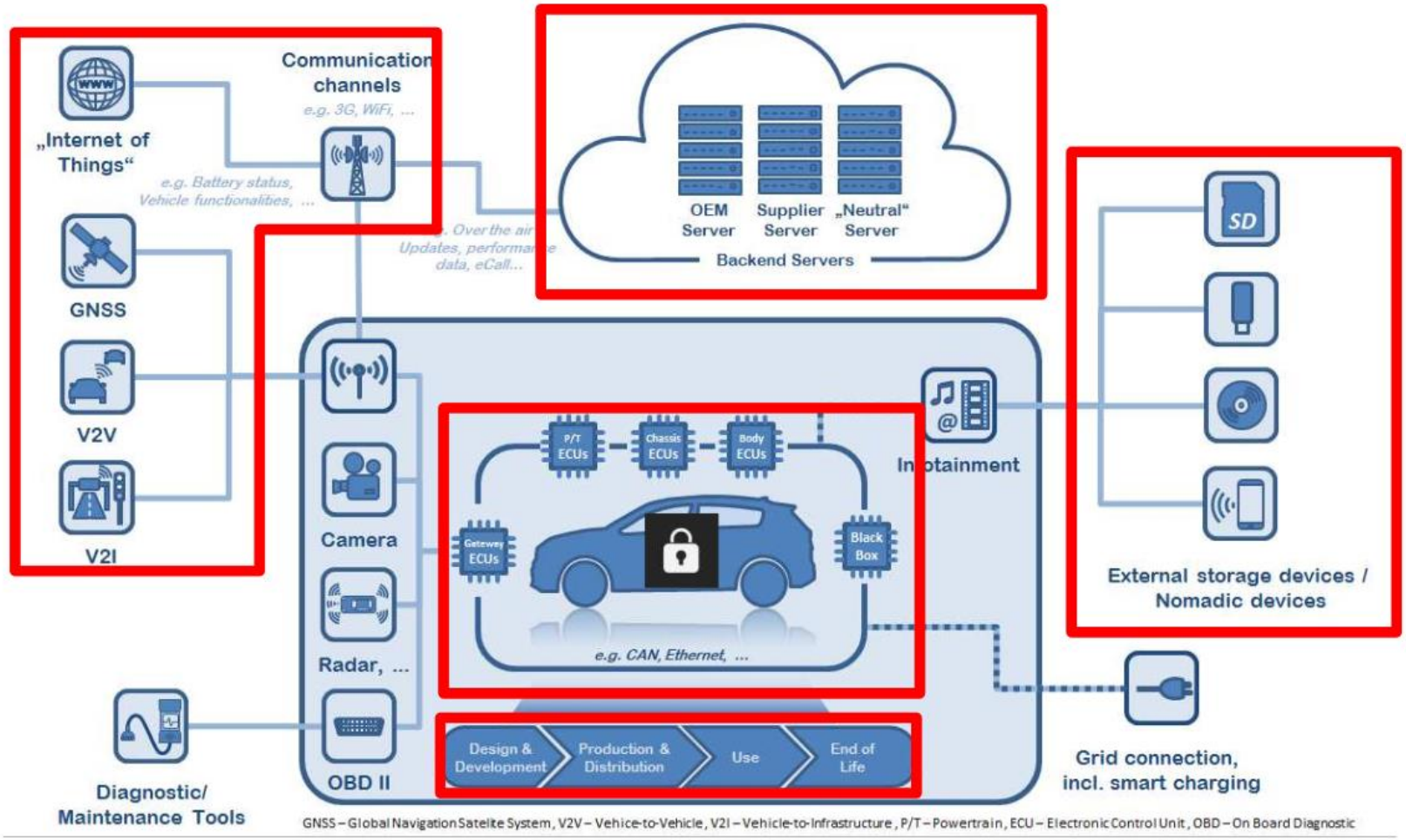
By Alistair Charlton

August 12, 2015 10:03 BST

ANDY GREENBERG SECURITY 09.27.16 7:00 AM

TESLA RESPONDS TO CHINESE HACK WITH A MAJOR SECURITY UPGRADE

Cyber Security (Many Considerations)



Cyber Security

- **Important consideration for designing AS**
- Have been **a number of attacks at car systems and sensors (e.g. LIDAR and GPS) that were used to manipulate the cars behavior**
- **Questions to ask:**
  1. Should there be a minimum security threshold to allow the system to be used?
  2. How secure must the systems and the connections be?
  3. Necessary to involve black boxes (for AV case)?
  4. Constant software updates to ensure cyber security?
  5. Necessary for connectivity functionalities?

Cyber Security

Possible Solutions:

1. DREAD (Risk Assessment Model)

- Damage – how bad would an attack be?
- Reproducibility – how easy is it to reproduce the attack?
- Exploitability – how much work is it to launch the attack?
- Affected users – how many people will be impacted?
- Discoverability – how easy is it to discover the threat?

2. STRIDE (Security Checklist)

	Threat	Property Violated	Threat Definition
S	Spoofing identify	Authentication	Pretending to be something or someone other than yourself
T	Tampering with data	Integrity	Modifying something on disk, network, memory, or elsewhere
R	Repudiation	Non-repudiation	Claiming that you didn't do something or were not responsible; can be honest or false
I	Information disclosure	Confidentiality	Providing information to someone not authorized to access it
D	Denial of service	Availability	Exhausting resources needed to provide service
E	Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

3. RELIABILITY

Reliability

- **Questions to ask:**
  1. How reliable is the cell network?
  2. What if there is no mobile network available?
  3. What if sensor(s) fail?
  4. Should there be **redundancy** for everything?
  5. Is there a threshold that determines when the system is reliable (e.g. when two out of four sensors fail)?
- **Different levels that should be considered for reliability:**
  1. Firstly, the **diagnostic of the system** that might be subject to failures
  2. Then, the **system's sensors** that enable the system to sense its surrounding environment
  3. Lastly, the **data coming from external entities**, like other vehicles and road infrastructures for AV case

4. RESPONSIBILITY & ACCOUNTABILITY

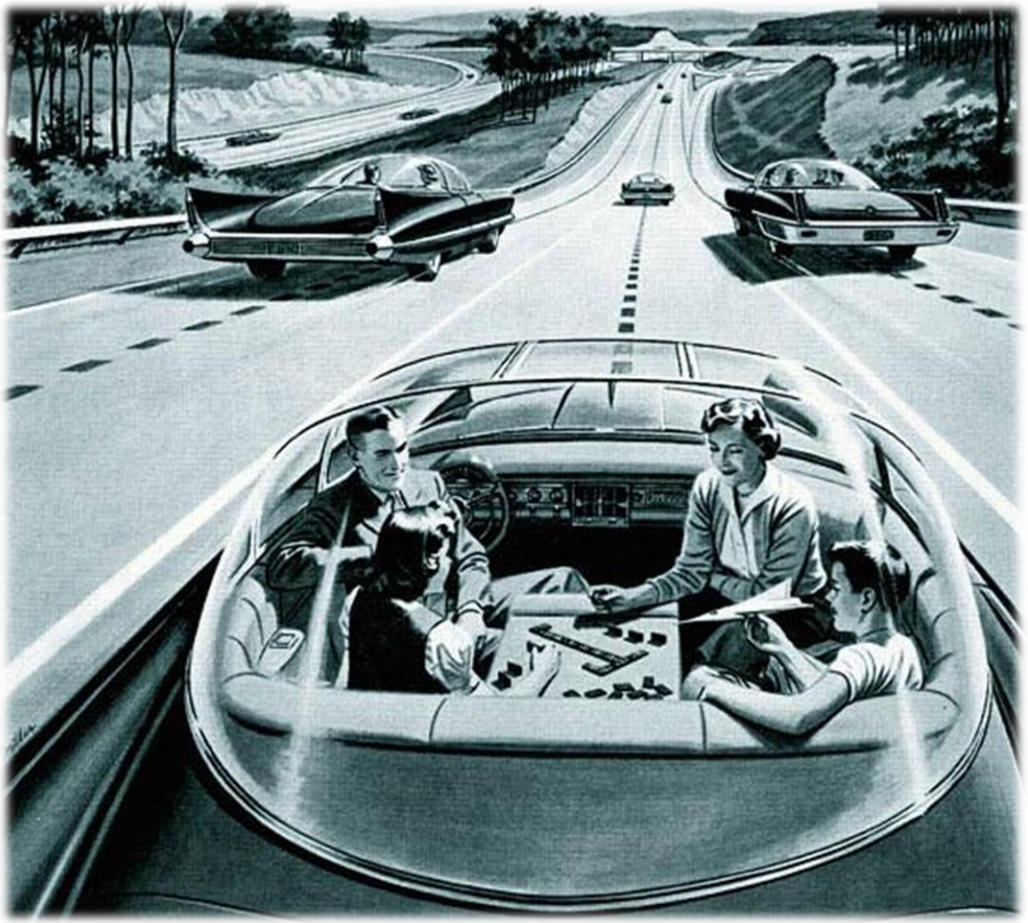
Responsibility & Accountability

- **How will responsibility be defined in case of incidents and accidents?**
- Should the **designer and/or manufacturer be held responsible?**
- Responsible design to **strictly adhere to standards or beyond standard requirements**
- Responsible design of embedded AI systems to **prevent abuse of AS** (e.g. drone attacks in Saudi Arabia)
- If the system is meant for weapon usage, responsible design to account for the **possibility of malfunction or rebellion** (e.g. terminator)

CHAPTER 4:
UNDERSTANDING THE CONCEPT OF TECHNOLOGIES AND SYSTEM ARCHITECTURES THAT ARE INVOLVED IN AV & AR

1. AUTONOMOUS VEHICLES

Concept of AV arise in 1956



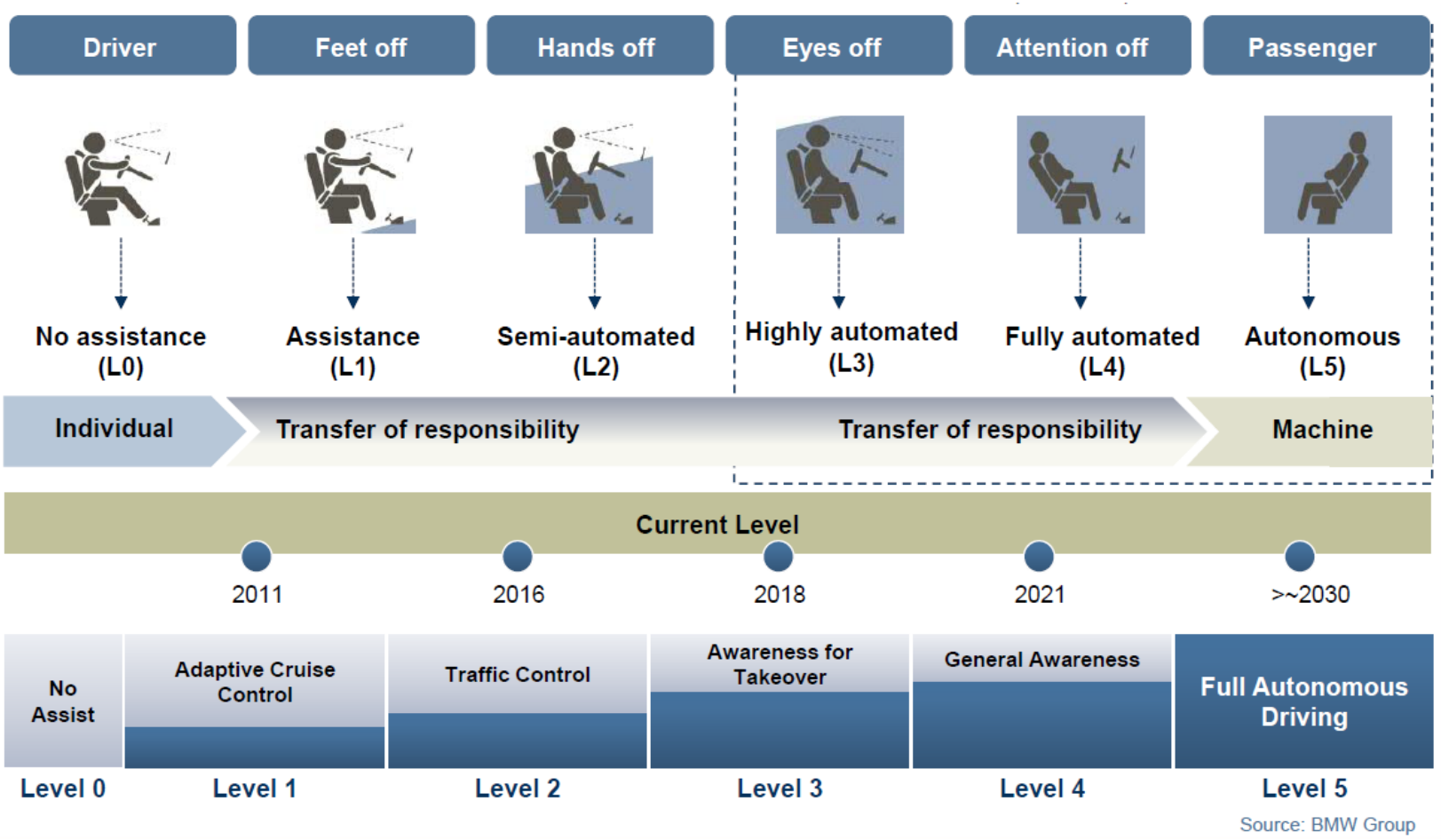
America's Power Companies' advertisement from 1956 depicting a future with autonomous cars

Driverless Waymo rides

Source: https://www.youtube.com/watch?v=x4jg4E7LrZE

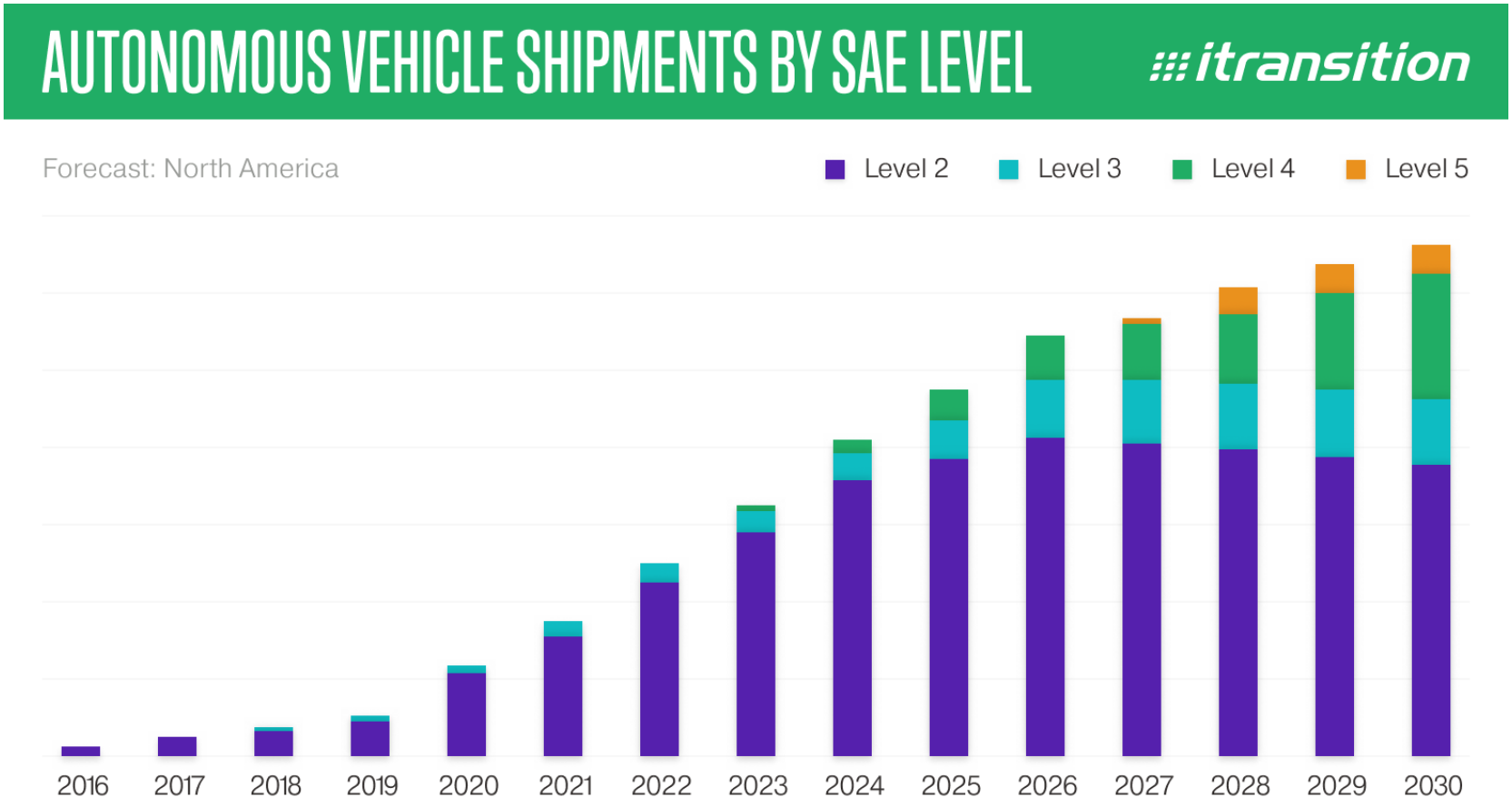
The 6 Levels of Autonomous Driving

The 6 Levels of Autonomous Driving

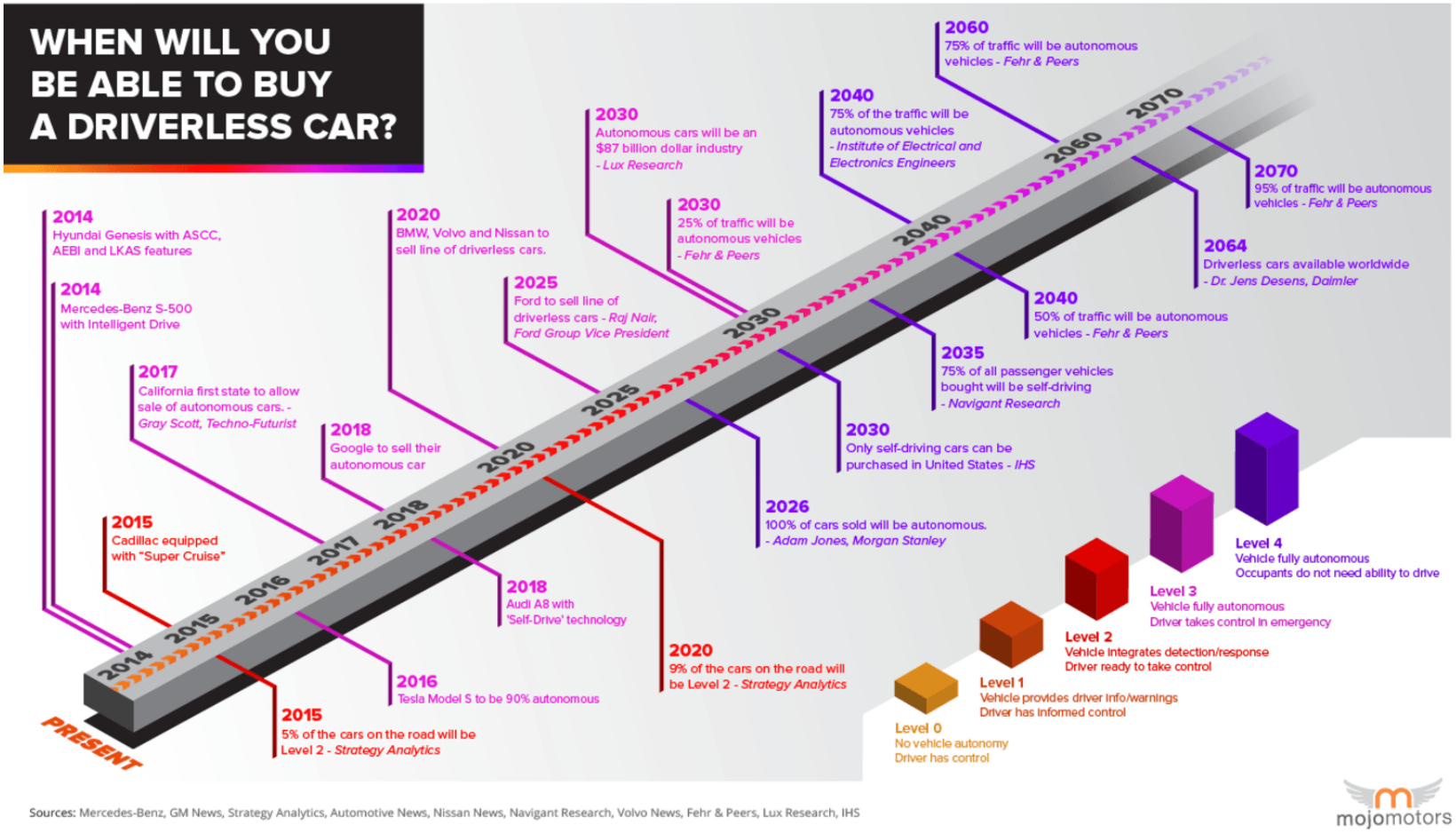


L5 Autonomous Driving Example:

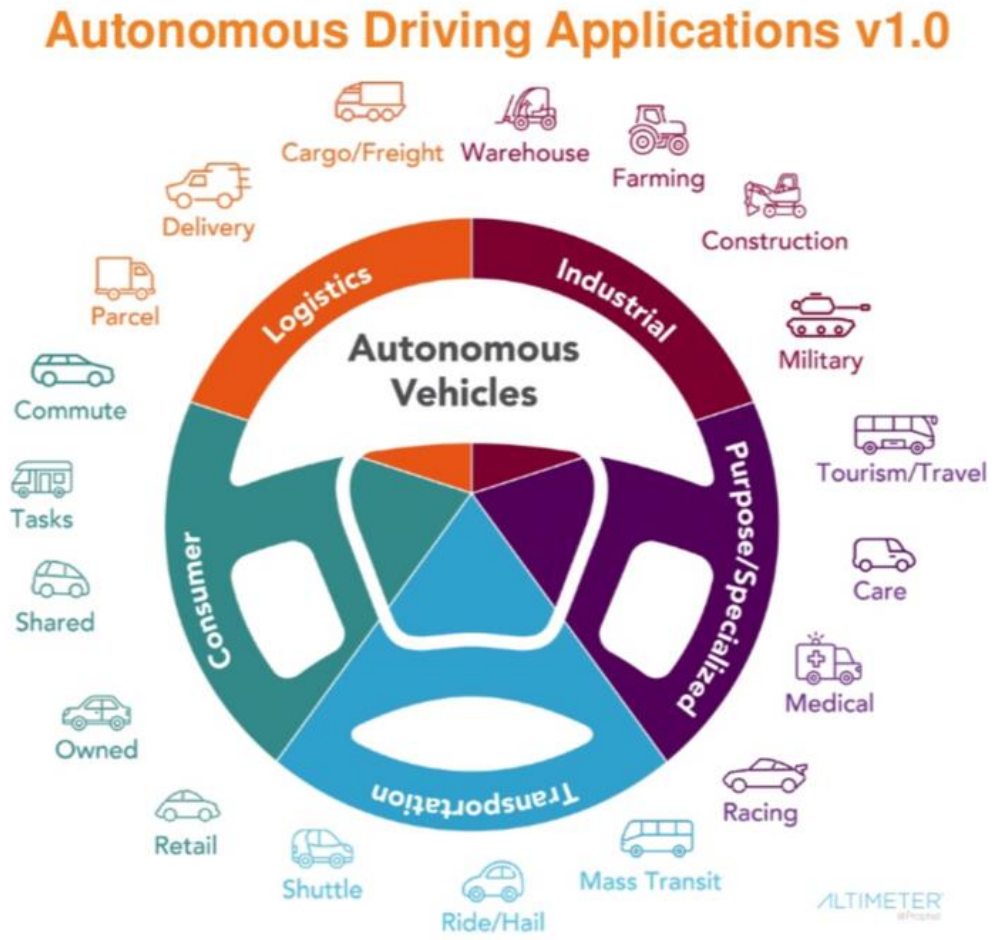
AV Technology – Are we there yet?



AV Projected Timeline



AV Applications



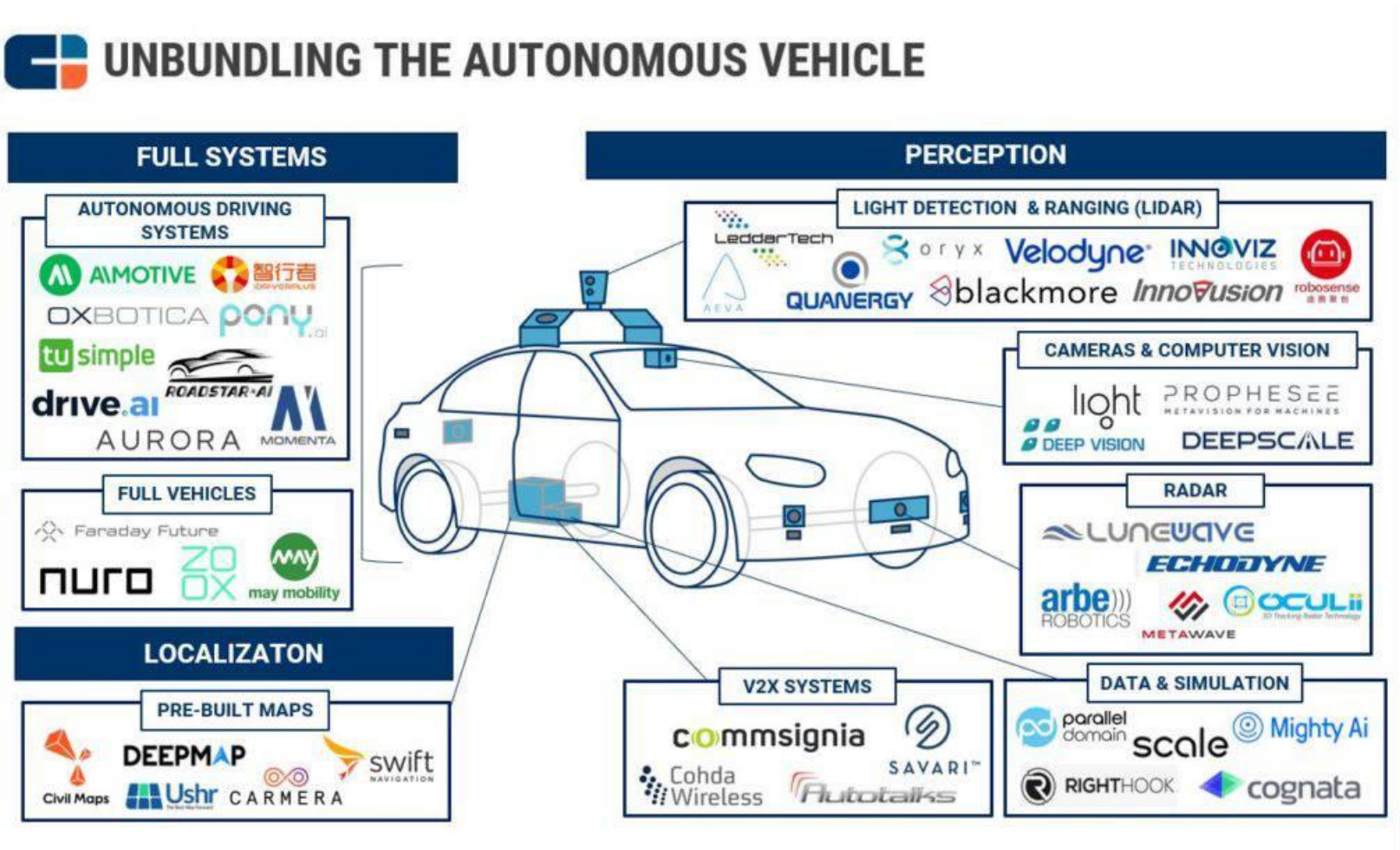
Current AV Players

Traffic Ahead

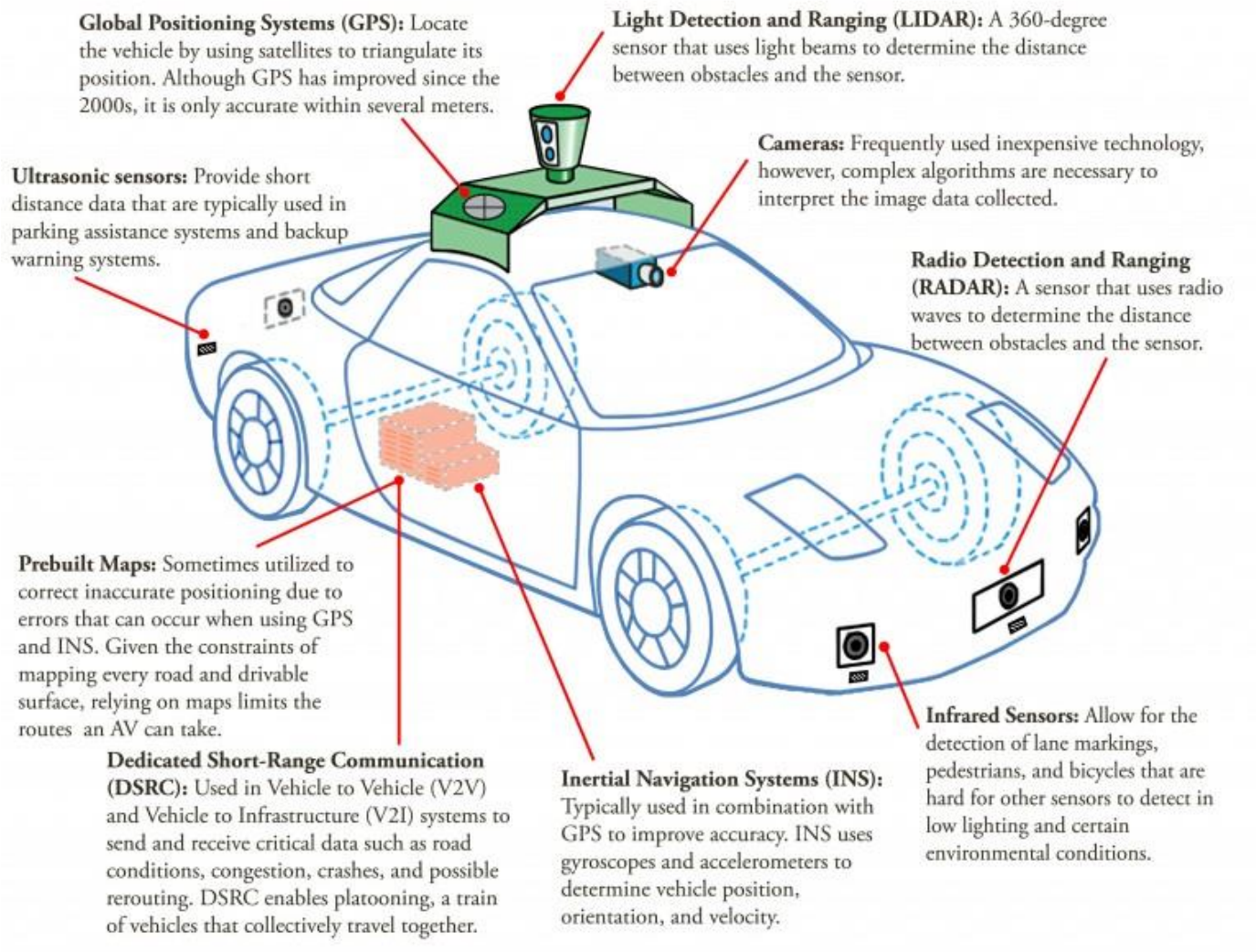
Many carmakers are developing prototype vehicles that are capable of driving autonomously in certain situations. The technology is likely to hit the road around 2020.

	 BMW	 Mercedes-Benz	 Nissan	 Google	 General Motors
VEHICLE	5 Series (modified)	S 500 Intelligent Drive Research Vehicle	Leaf EV (modified)	Prius and Lexus (modified)	Cadillac SRX (modified)
KEY TECHNOLOGIES	• Video camera tracks lane markings and reads road signs • Radar sensors detect objects ahead • Side laser scanners • Ultrasonic sensors • Differential GPS • Very accurate map	• Stereo camera sees objects ahead in 3-D • Additional cameras read road signs and detect traffic lights • Short- and long-range radar • Infrared camera • Ultrasonic sensors	• Front and side radar • Camera • Front, rear, and side laser scanners • Four wide-angle cameras show the driver the car's surroundings	• LIDAR on the roof detects objects around the car in 3-D • Camera helps detect objects • Front and side radar • Inertial measuring unit tracks position • Wheel encoder tracks movement • Very accurate map	• Several laser sensors • Radar • Differential GPS • Cameras • Very accurate map

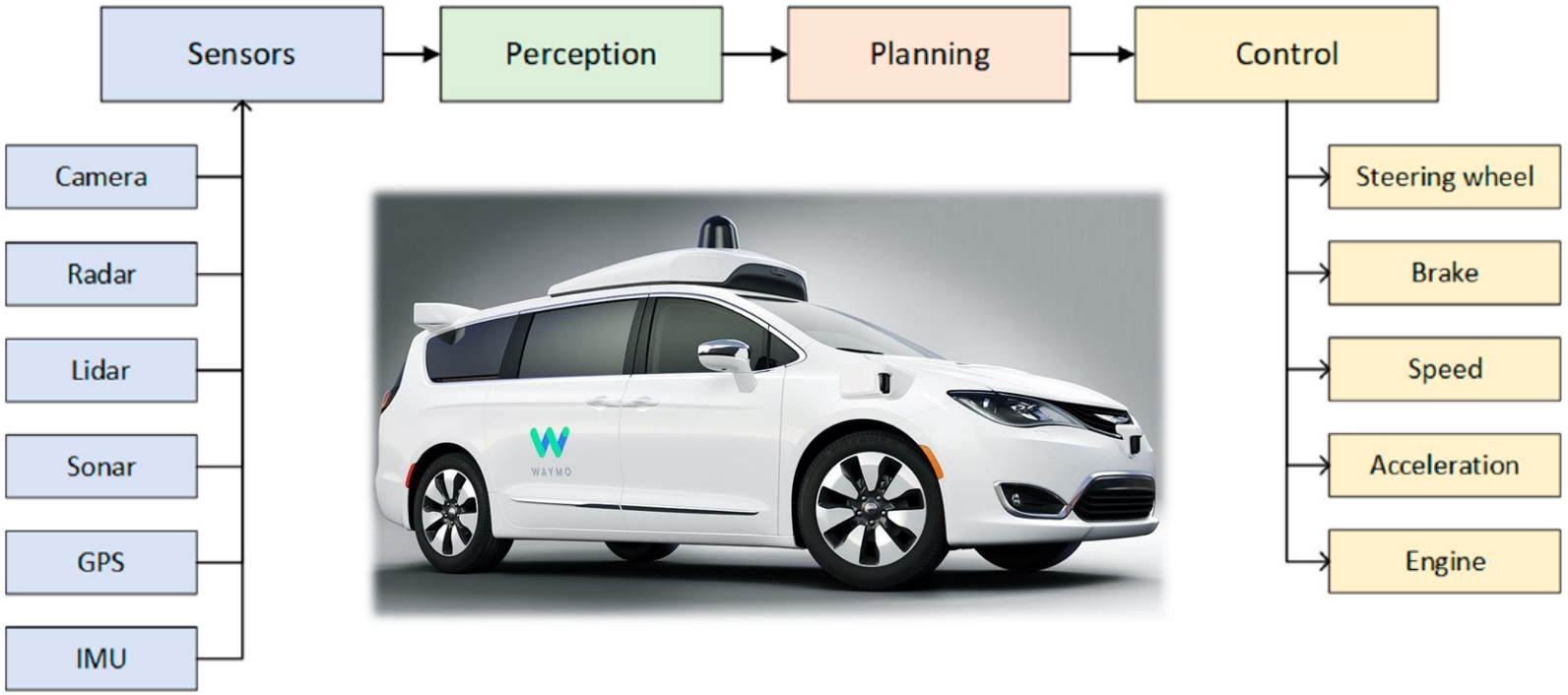
Current AV Players



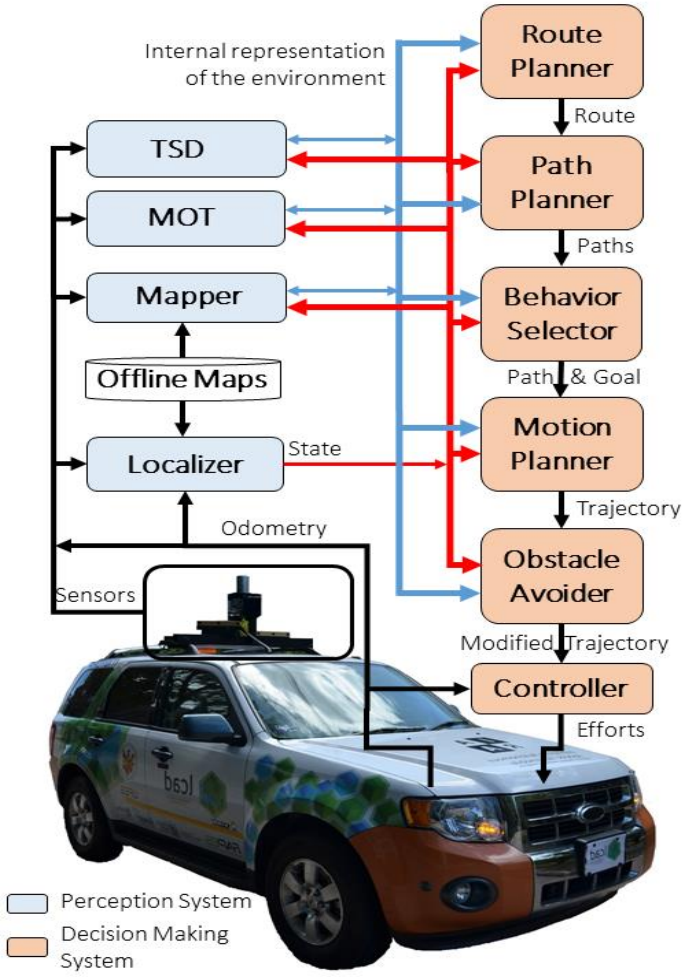
AV Technologies



AV System Block Diagram

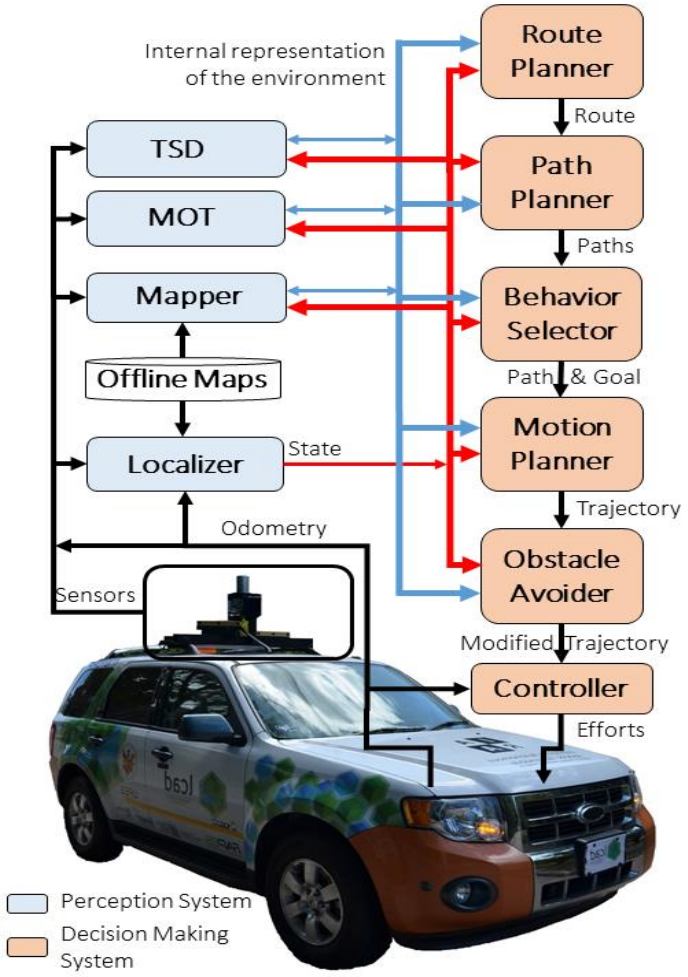


Typical Architecture of Self-Driving Cars



- **Mainly Perception & Decision-Making Systems**
- **Perception system** is responsible for:
  - **Estimating the State of the car**
  - **Creating an internal (to the self-driving system) representation of the environment,**
  - **Using data captured by on-board sensors**
    - E.g. Light Detection and Ranging (LIDAR), Radio Detection and Ranging (RADAR), camera, Global Positioning System (GPS), Inertial Measurement Unit (IMU), odometer
  - **And prior information about the sensors' models, road network, traffic rules, car dynamics**

Typical Architecture of Self-Driving Cars



- **Decision-Making system** is responsible for **navigating the car from its initial position to the final goal defined by the user,**
- **The current car's State** and **the internal representation of the environment,** as well as **traffic rules** and **passengers' safety and comfort are considered** during this process
- This system **needs to know where the car is in which location within the environment → State Data from Localizer**

Singapore's Progress: Autonomous Vehicles

Singapore's Progress: CETRAN

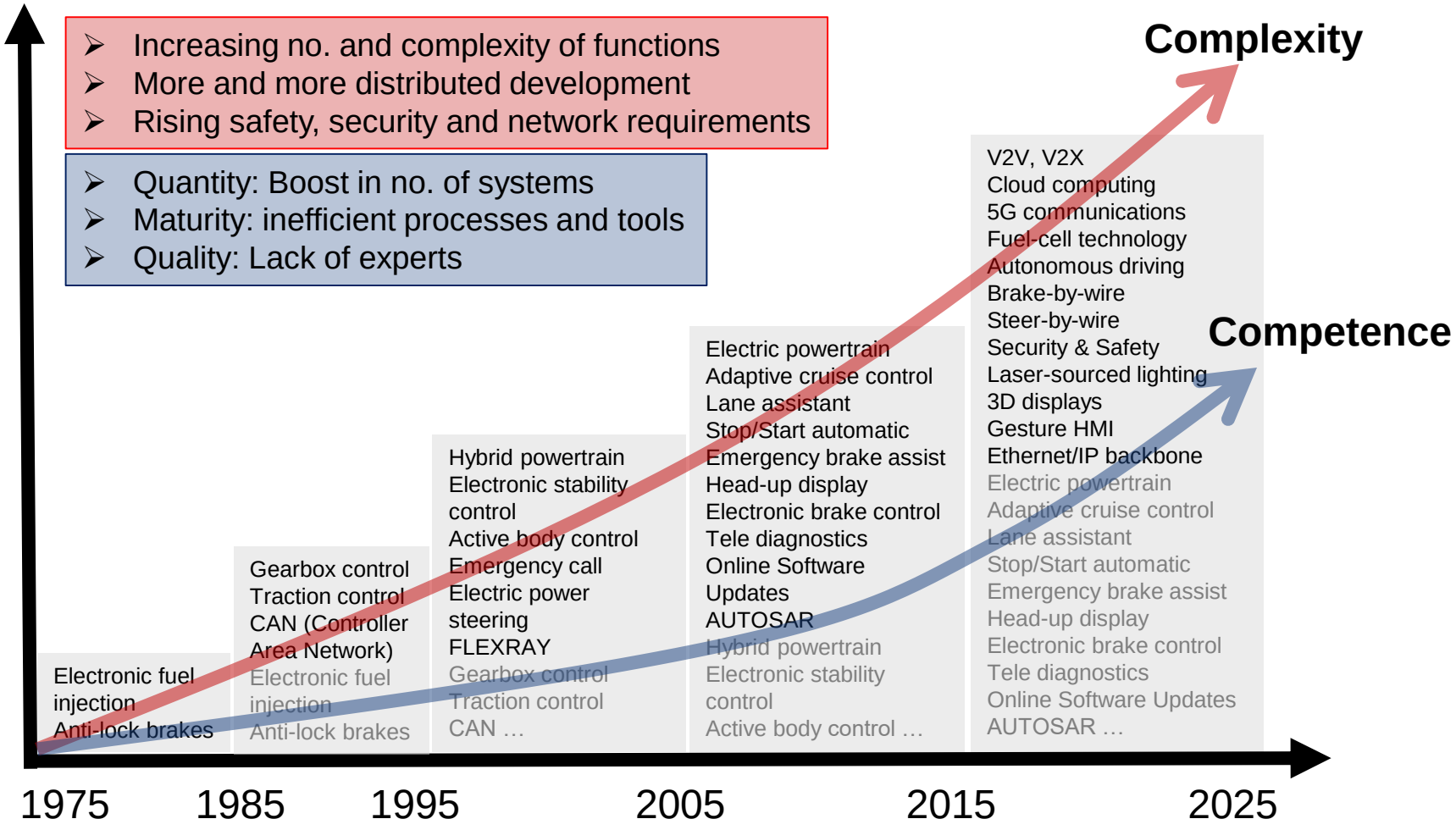
Singapore's first Autonomous Vehicle Test Centre

Singapore's Progress: Autonomous Electric Bus

First of two Volvo Electric buses will soon begin trials at the NTU Smart Campus before being extended to public roads

Nanyang Technological University, Singapore (NTU Singapore) and Volvo Buses have launched the world's **first full size, autonomous electric bus** today. The single-deck Volvo Electric bus is 12 metres long and has a full capacity close to 80 passengers.

Exploding complexity of AVs

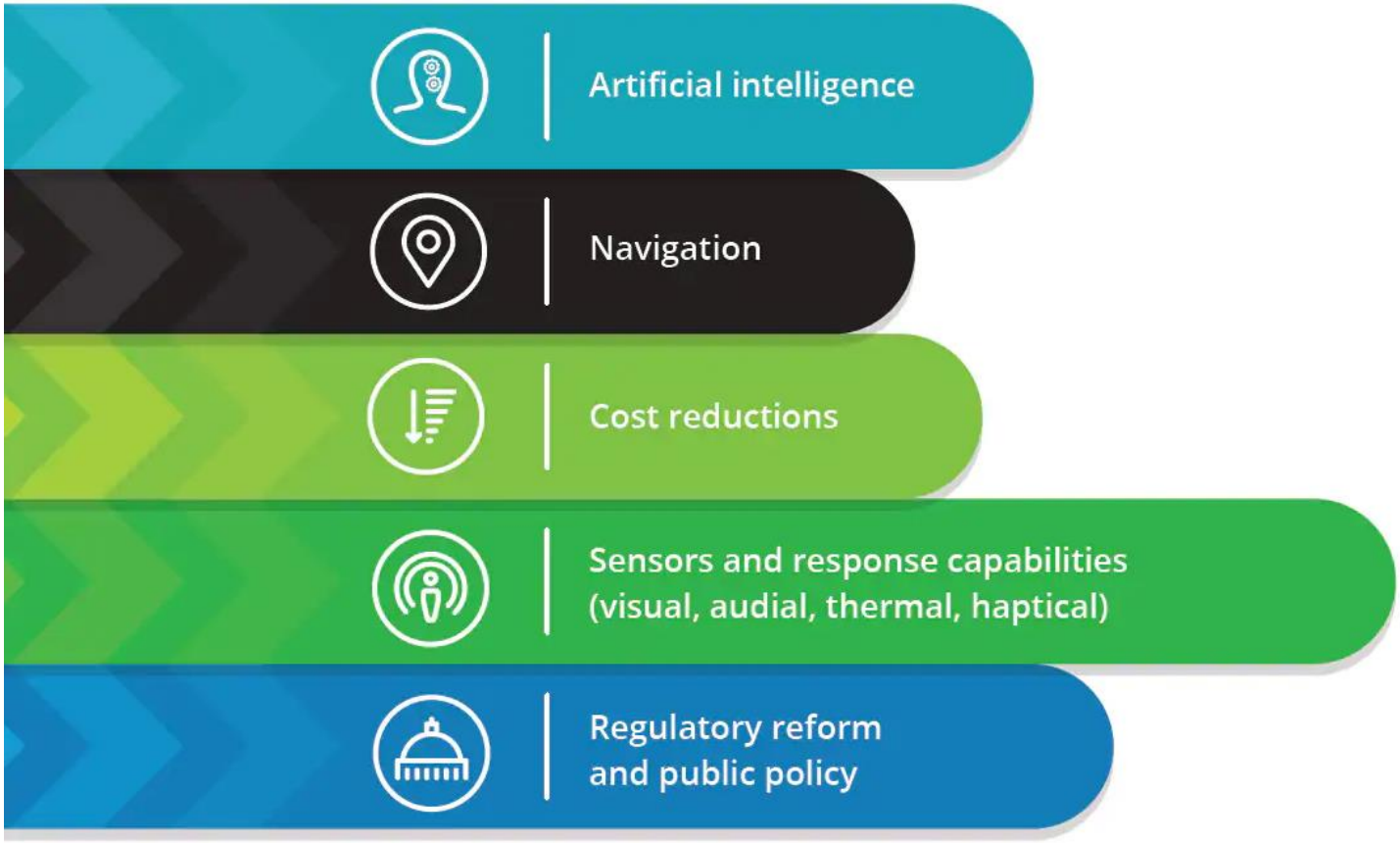


2. AUTONOMOUS ROBOTS

Nimbo - Segway's Autonomous Patrols

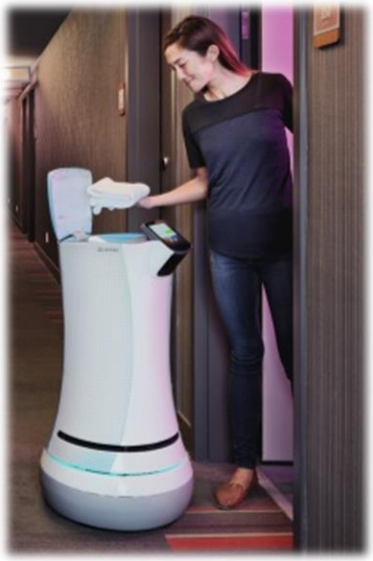
Source: https://www.youtube.com/watch?v=Mog3UgQvHIo

5 Key Developments in Autonomous Robots



Types of Robots based on their control system

Autonomous controlled robots

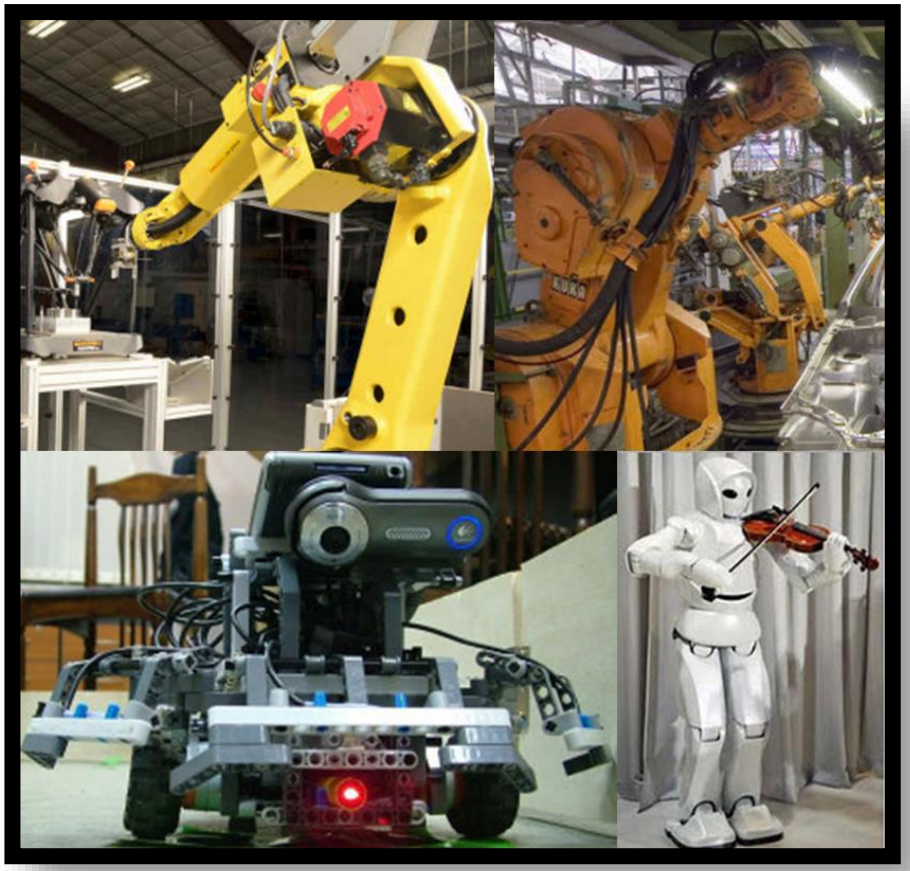


Semi-Autonomous controlled robots

Manually controlled robots

Types of Autonomous Robots

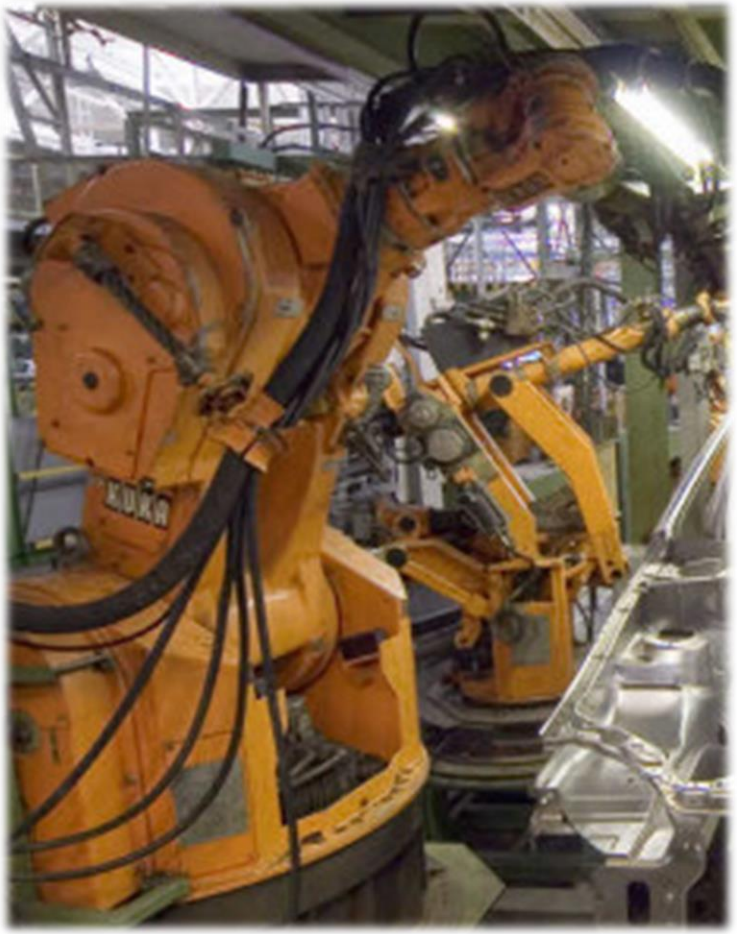
1. Non-programmable
2. Programmable
3. Adaptive
4. Intelligent



Types of Autonomous Robots

1. Non-programmable AR

- Basic type of robot
- An exploiter lacking reprogrammable controlling device
- Non-versatile; unable to change function and application
- Mostly used in industries for mass production purposes



Types of Autonomous Robots

2. Programmable AR

- Reprogrammable and versatile
- Function and application can be changed
- Perform function in the given pattern and fixed sequence
- Drawback: Once programmed it persists operation even if there is a need to change its task (e.g. in the event of emergency)

Types of Autonomous Robots

3. Adaptive AR

- Can be adapted independently to various environments
- More sophisticated than programmable robots
- Can be adapted up to a certain extent
- Utilize sensors and control systems

Types of Autonomous Robots

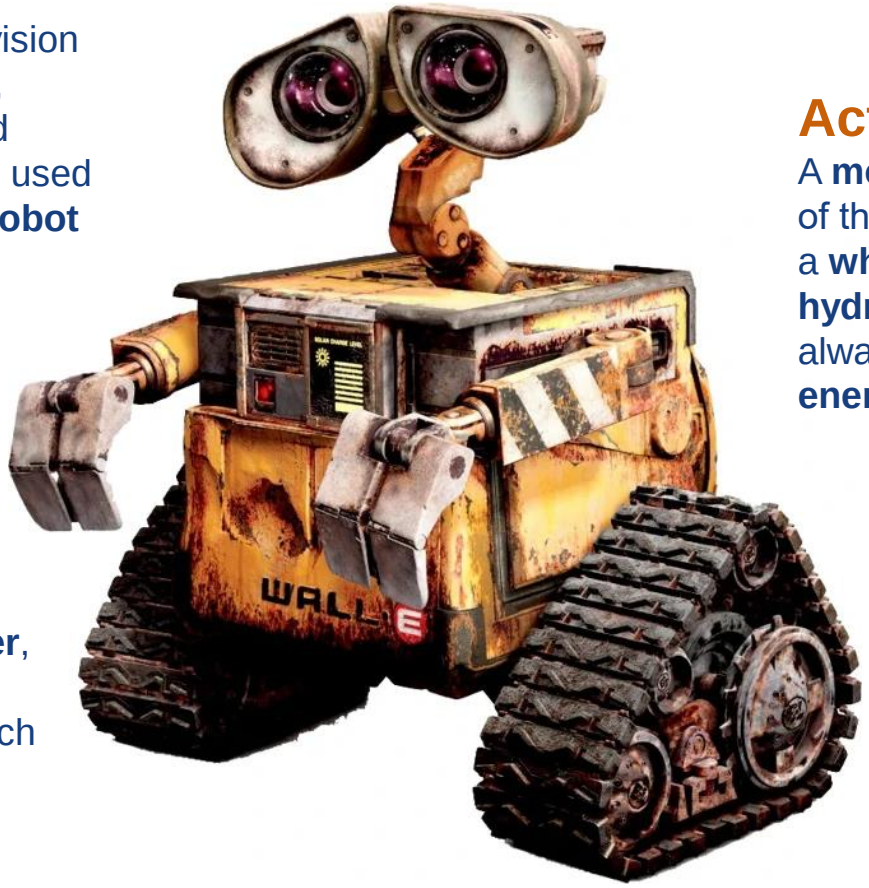
4. Intelligent AR

- Most intelligent out of all
- Utilize sensors and microprocessors
- Utilize AI algorithms (e.g. Knowledge reasoning, planning, deep learning, machine vision)
- Highly efficient because of their situation-based analyzing and task performing abilities

AR Technologies

Perception

Laser scanners, stereo vision cameras, bump sensors, force-torque sensors and spectrometers can all be used as **input devices for a robot**



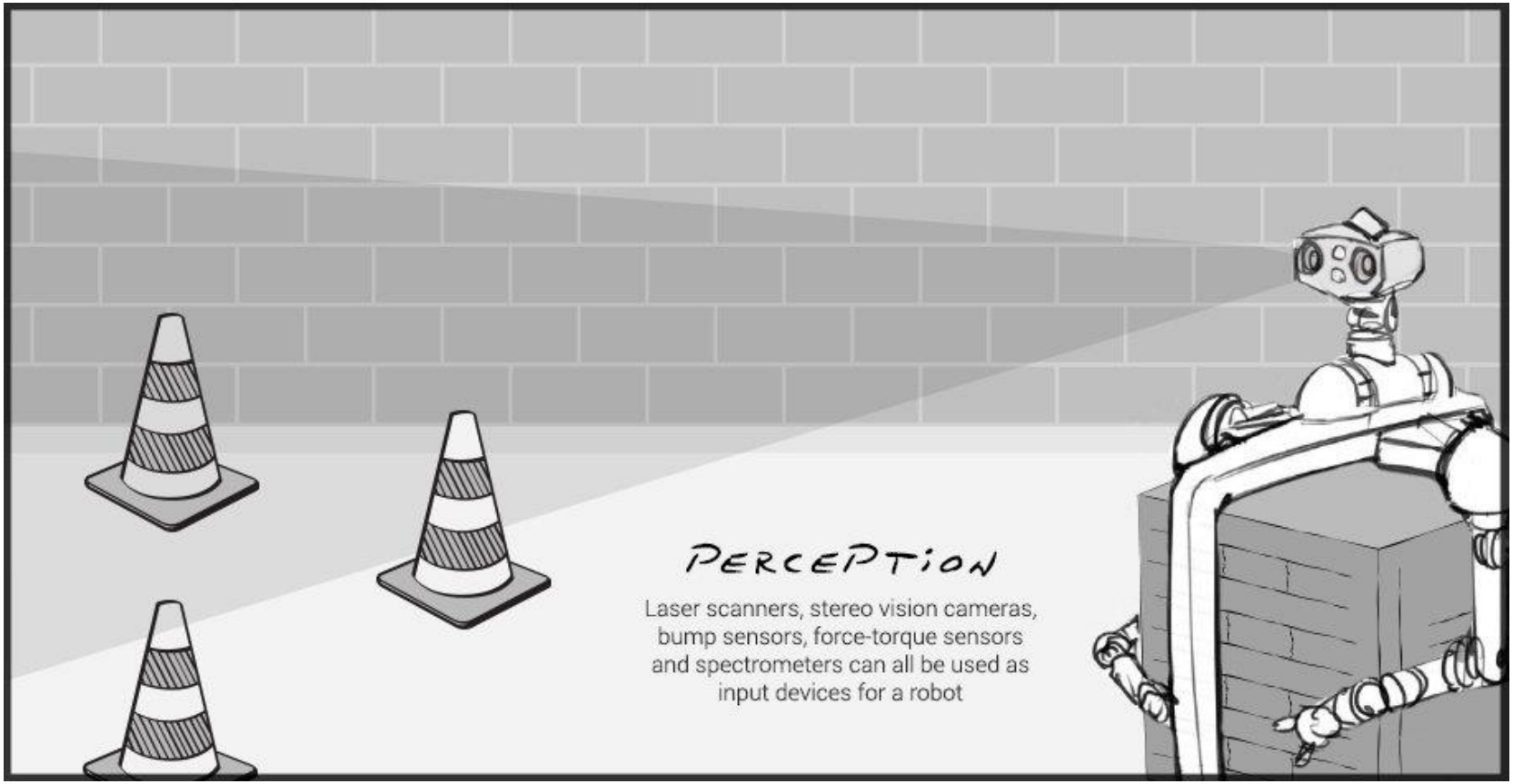
Actuation

A **motor** is usually the heart of the actuator. Whether it's a **wheel, linear actuator or hydraulic ram**, there's always a motor **converting energy into movement**

Decision

In addition to a **computer**, the robot utilizes an **embedded system**, which operates faster and with higher authority than the computer

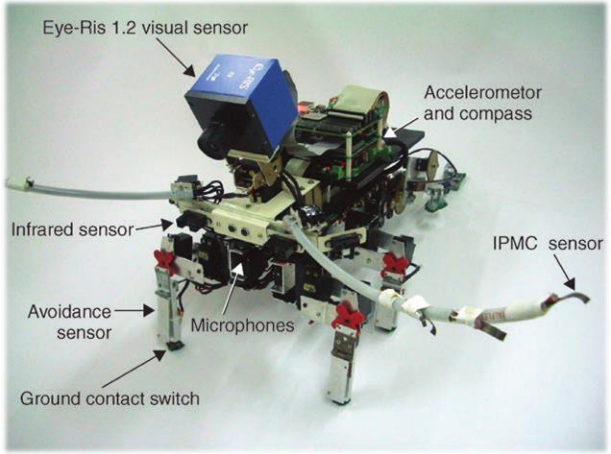
Robotic Perception



Robotic Perception

Robot Sensory Systems (Examples)

Sense:	Possible Relevant Sensor:
• **Vision (Sight)**	• **Camera**
• Audition (Hear)	• Microphone
• Gustation (Taste)	• Chemical sensors
• Olfaction (Smell)	• Chemical sensors
• Tactitions (Feel)	• Contact sensors
• Thermoception (Heat)	• Thermocouple

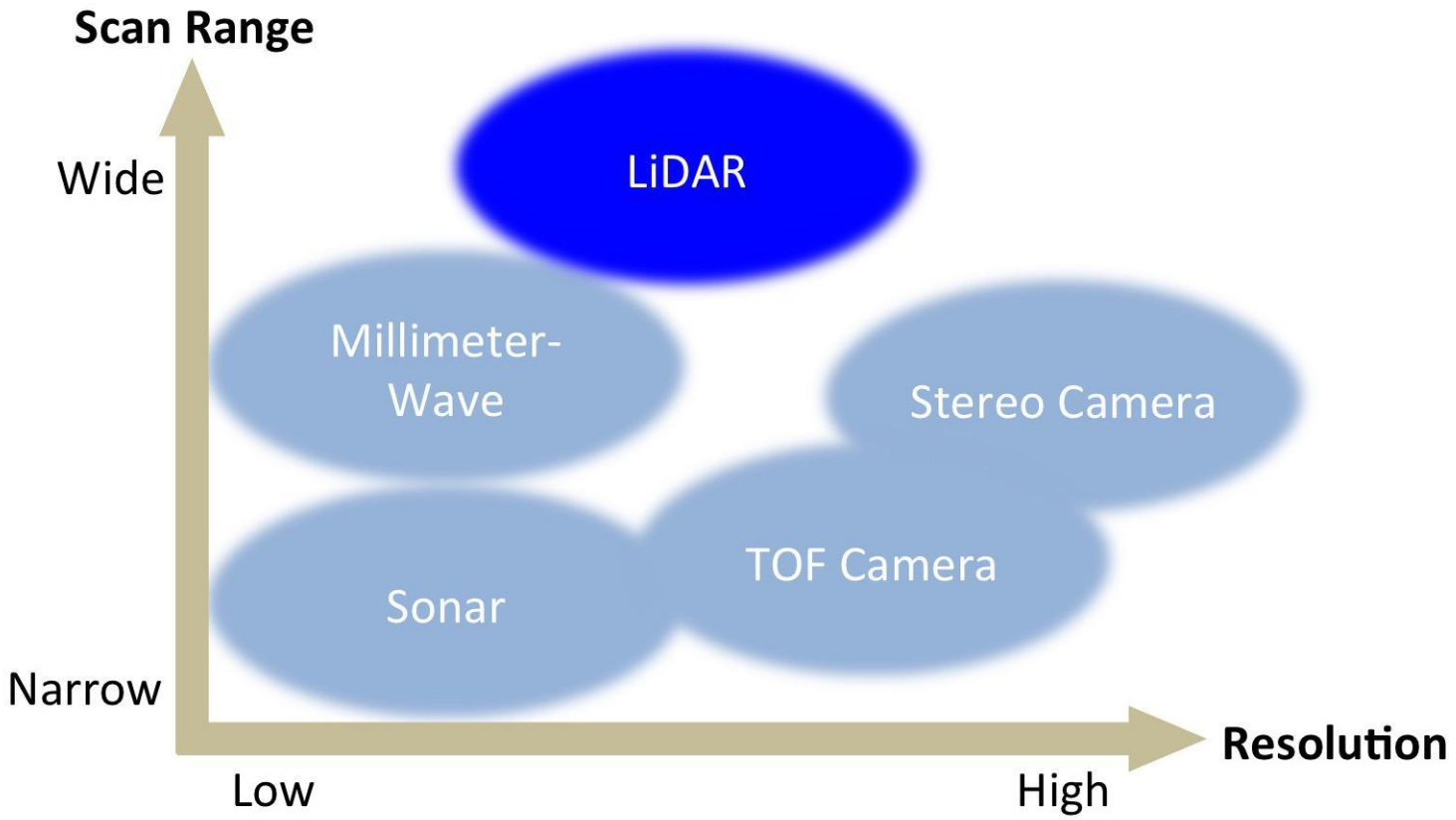


Robot Sensory Systems (Examples)

Sense:	Possible Relevant Sensor:
• Equilibrioception	• Accelerometer
• Proprioception	• Encoders
• Magnetoception	• Magnetometer
• Electroception	• Voltage sensor
• Echolocation	• Sonar
• Pressure gradient	• Pressure sensors

Robotic Perception

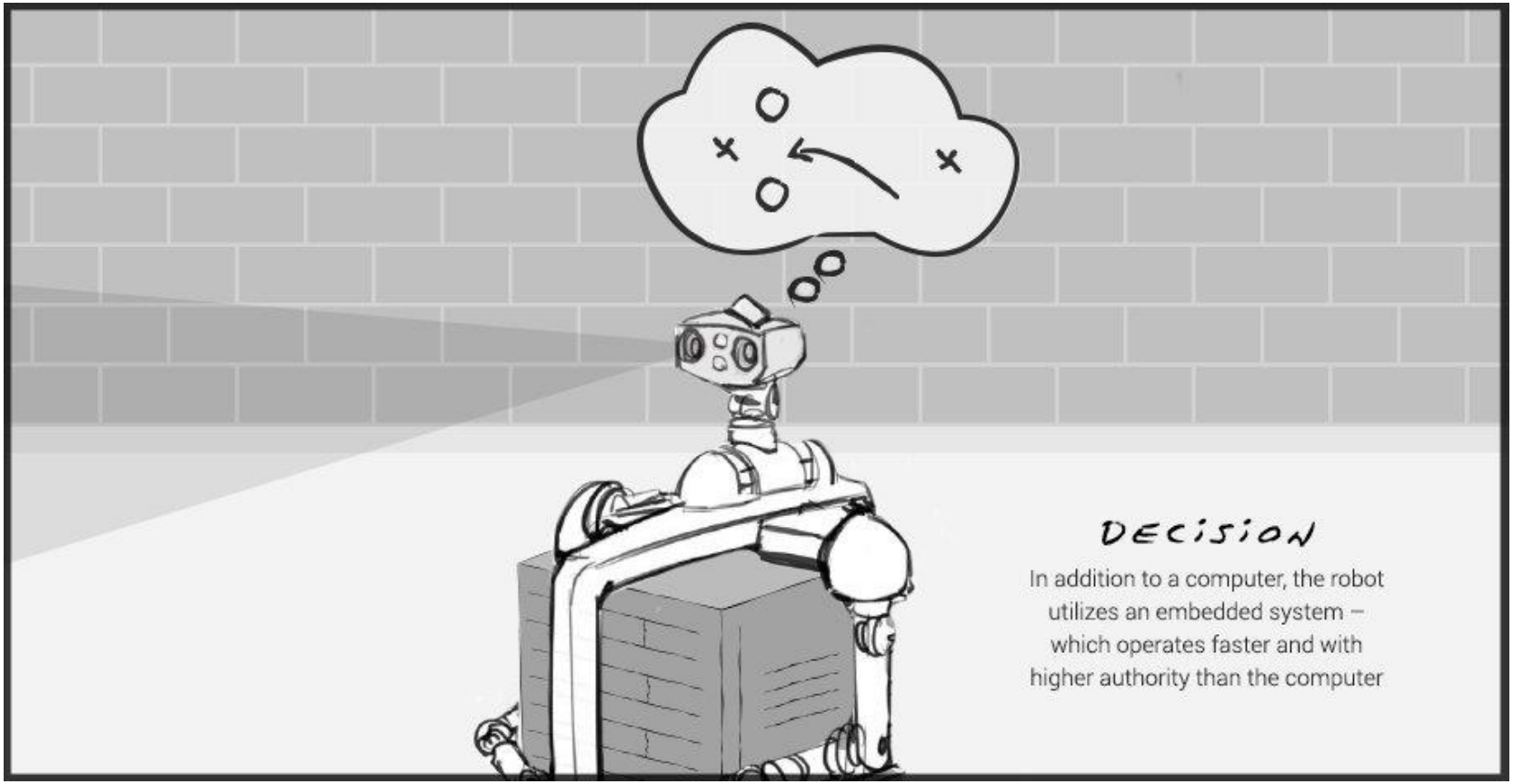
Autonomous robots need a wide angle of view



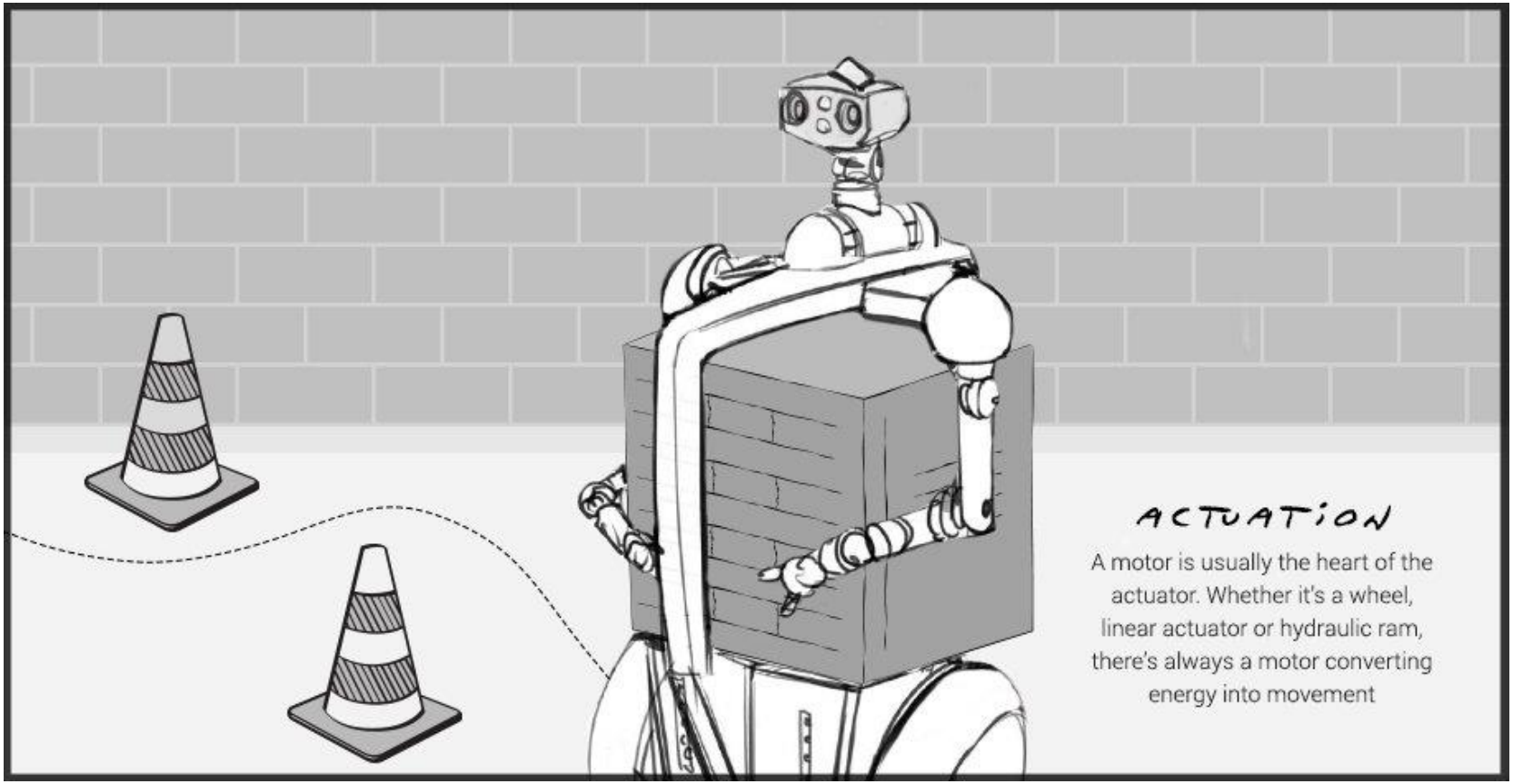
Data Sensor Fusion

- Defined as: **The process of integrating together different sensory data from multiple sources into the same representation framework**
- **Example: Xiaomi Mi Robot;** Equipped with **12 different sensors** that prevent the robot from bumping into things and falling down (off the stairs)

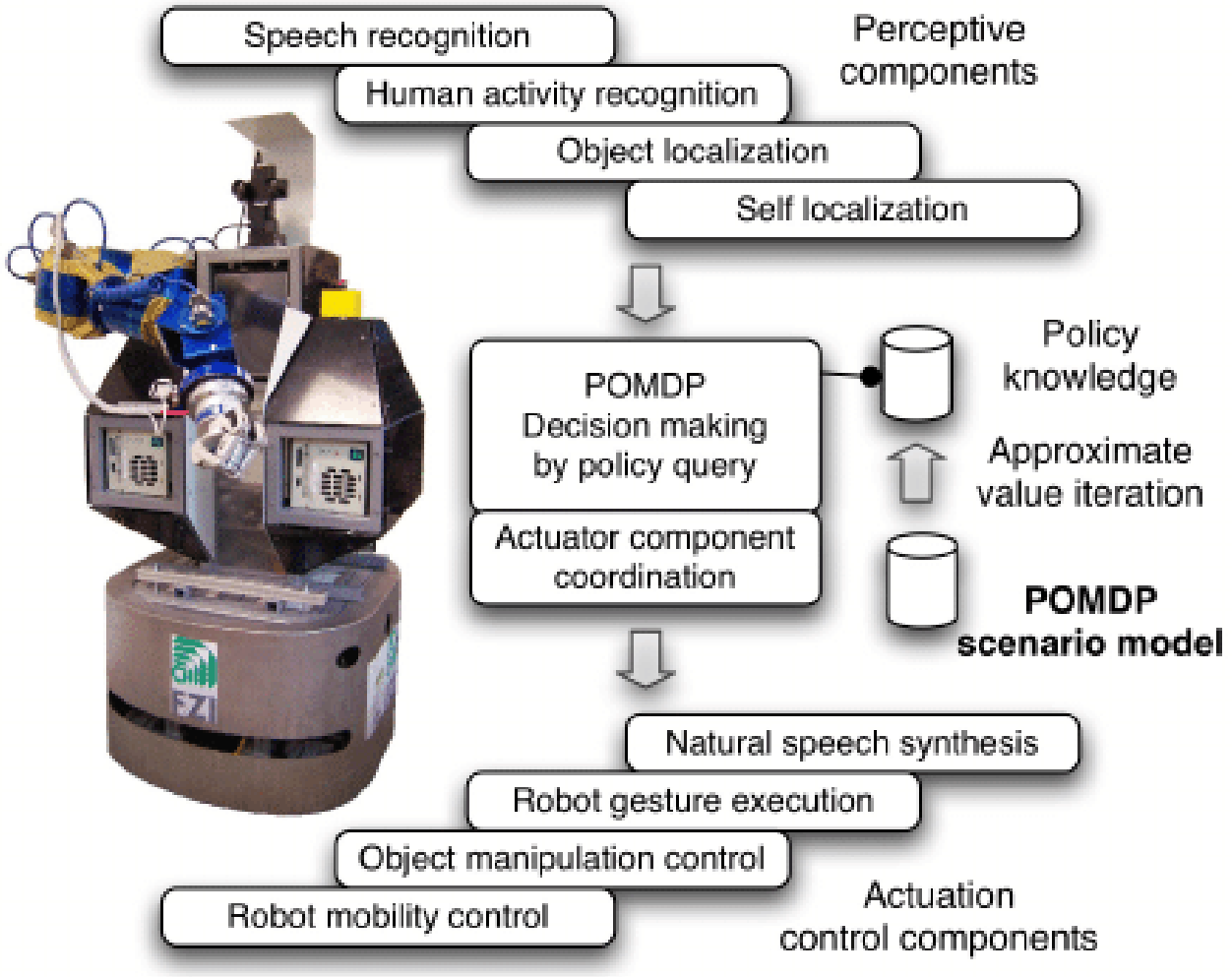
Robotic Decision



Robotic Actuation



Typical Architecture of Autonomous Robots



Autonomous Robots Applications

RoboCup Competition:

Source: https://youtu.be/_PC-V5GJP6Q

WORKSHOP: DEVELOP BASIC AUTONOMOUS ROBOT ARM

1. INTRODUCTION TO ROS, TURTLEBOT3 & OPEN MANIPULATOR-X

Download & Install Autonomous ROS Package

Steps:

1. Change to the source space directory of the catkin workspace:

```
$ cd ~/catkin_ws/src
```

2. Git Clone the Autonomous package:

```
$ git clone
https://github.com/nicholashojunhui/autonomous.git
```

3. Build the packages in the catkin workspace:

```
$ cd ~/catkin_ws && catkin_make
```

4. Go to *catkin_ws/src/autonomous/src* and make all python files executable

Creating a new ROS Package (Optional)

Steps:

1. Change to the source space directory of the catkin workspace:

```
$ cd ~/catkin_ws/src
```

2. Use the catkin_create_pkg script to create a new package called '***new_package***' which depends on std_msgs, roscpp, and rospy:

```
$ catkin_create_pkg new_package std_msgs rospy roscpp
```

3. Build the packages in the catkin workspace:

```
$ cd ~/catkin_ws
$ catkin_make
```

4. Add the workspace to your ROS environment by sourcing the generated setup file:

```
$ . ~/catkin_ws/devel/setup.bash
```

Roslaunch vs Rosrun RECAP

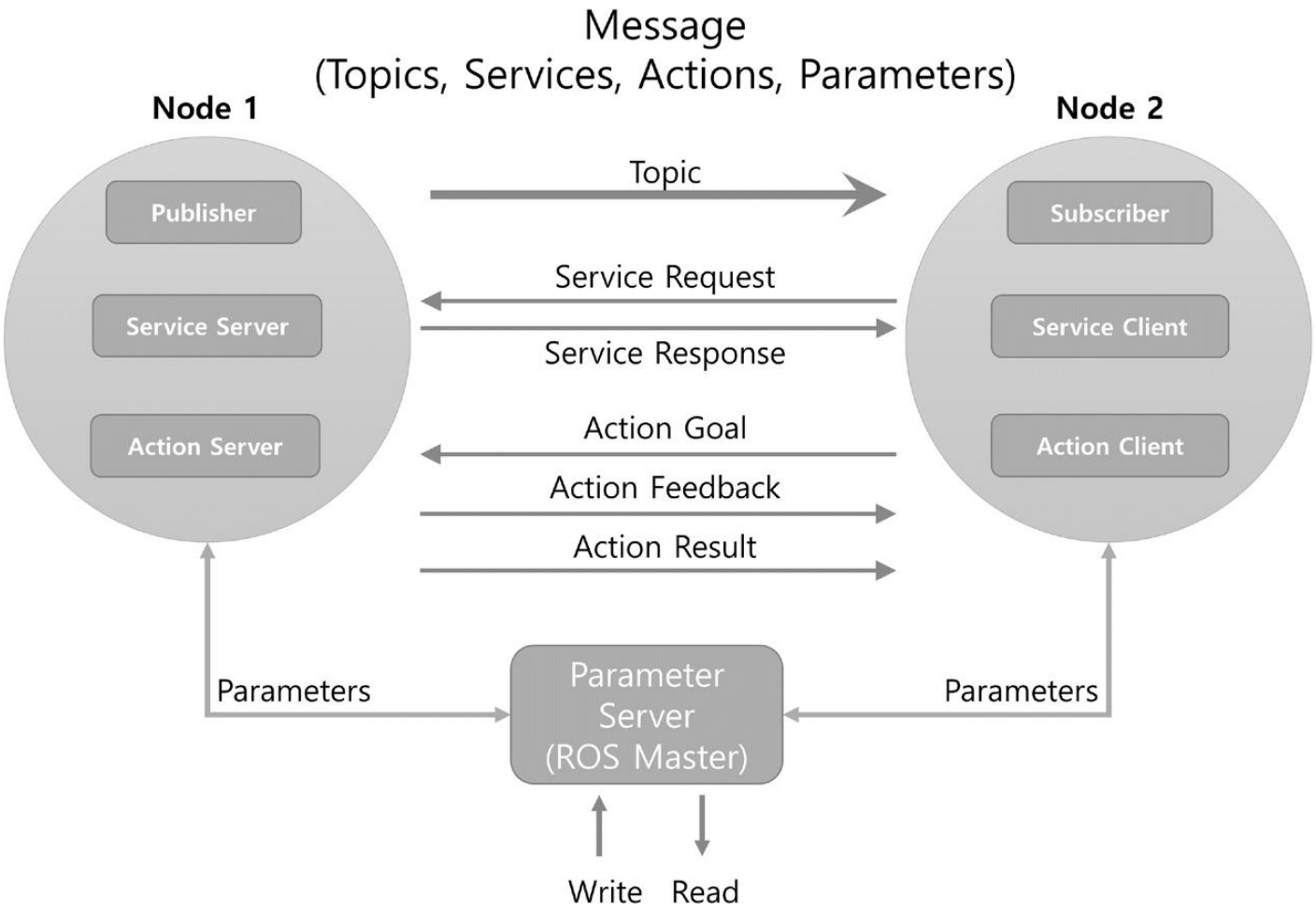
- **Rosrun** is a command that runs only one node in the specified package

```
rosrun [PACKAGE_NAME] [NODE_NAME]
```

- **Roslaunch** is a command that executes more than one node in the specified package or sets execution options

```
roslaunch [PACKAGE_NAME] [launch_FILE_NAME]
```

ROS Communication Block Diagram RECAP:



Introducing TurtleBot3

Source: https://youtu.be/9OC3J53RUsk

Introducing TurtleBot3

- A **ROS standard platform robot**
- Adopts ROBOTIS smart actuator Dynamixel for driving
- **Small, affordable, programmable** mobile robot
- Evolved with cost-effective and small-sized Single Board Computer (SBC) that is suitable for robust embedded system, 360 degree distance sensor and 3D printing technology
- Core technology is **SLAM** (simultaneous localization and mapping), **Navigation and Manipulation**
- Able to configure **high levels of autonomy**

Introducing TurtleBot3

ISO 29990:2010

WORLD'S MOST POPULAR ROS PLATFORM

TurtleBot is the world's most popular open source robot for education and research.

AFFORDABLE COST

TurtleBot is the most affordable platform for educations and prototype research & developments.

SMALL SIZE

Imagine the TurtleBot in your backpack and bring it anywhere.

EXTENSIBILITY

Extend ideas beyond imagination with various SBC, sensor, motor and flexible structure.

MODULAR ACTUATOR

Easy to assemble, maintain, replace and reconfigure.

OPEN SOURCE SOFTWARE

Variety of open source software for the user.
You can modify downloaded source code and share it with your friends.

OPEN SOURCE HARDWARE

Schematics, PCB Gerber, BOM and 3D CAD data are fully opened to the user.

STRONG SENSOR LINEUPS

8MP Camera, Enhanced 360° LiDAR,
9-Axis Inertial Measurement Unit and precise encoder for your robot.

TurtleBot3 Variations

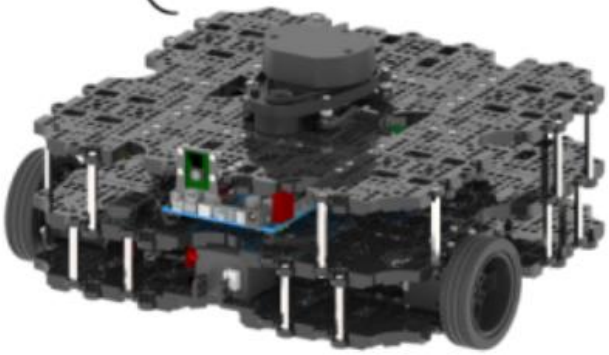
TURTLEBOT3

TurtleBot3
Burger

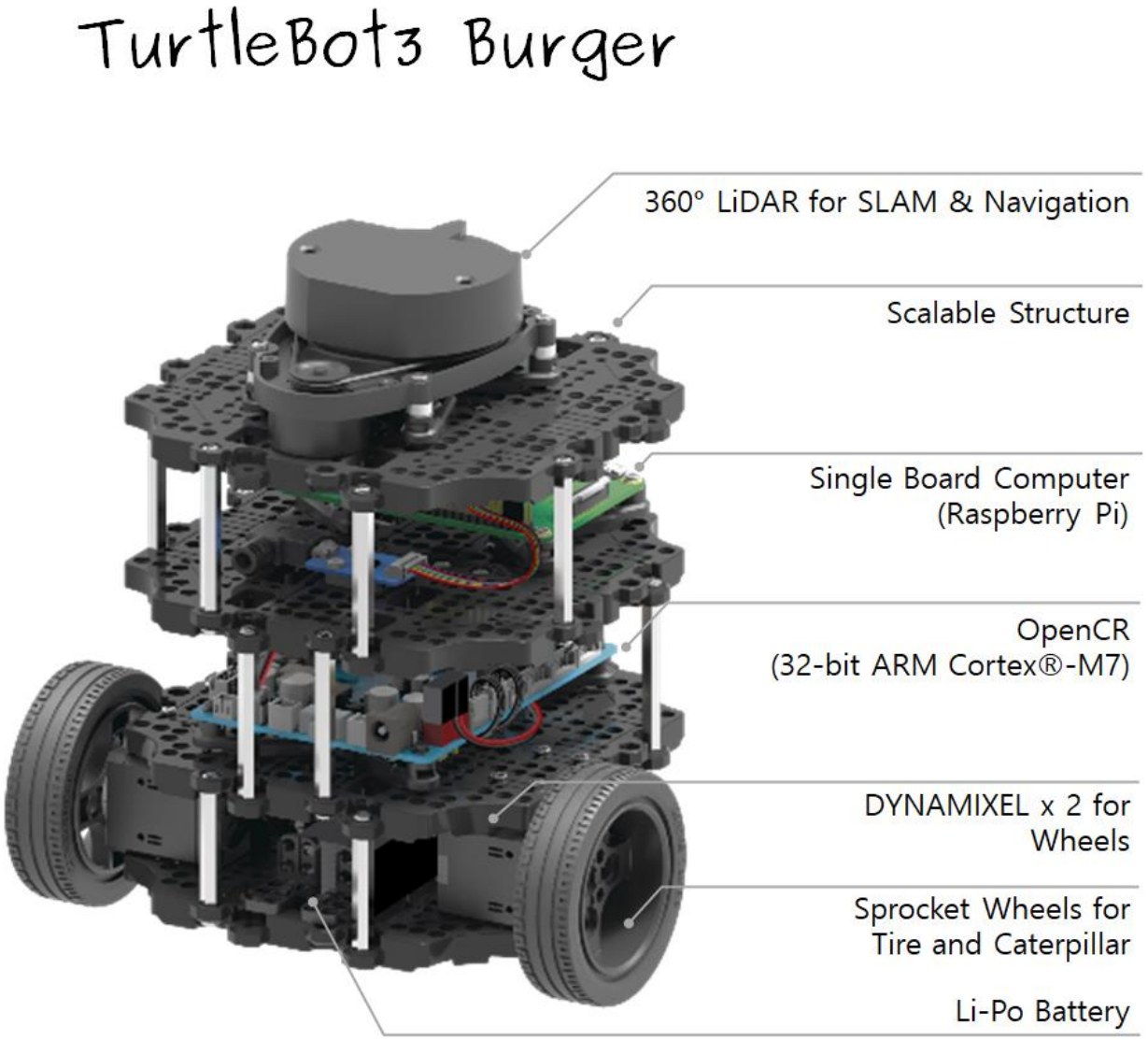
TurtleBot3
Waffle

Discontinued!

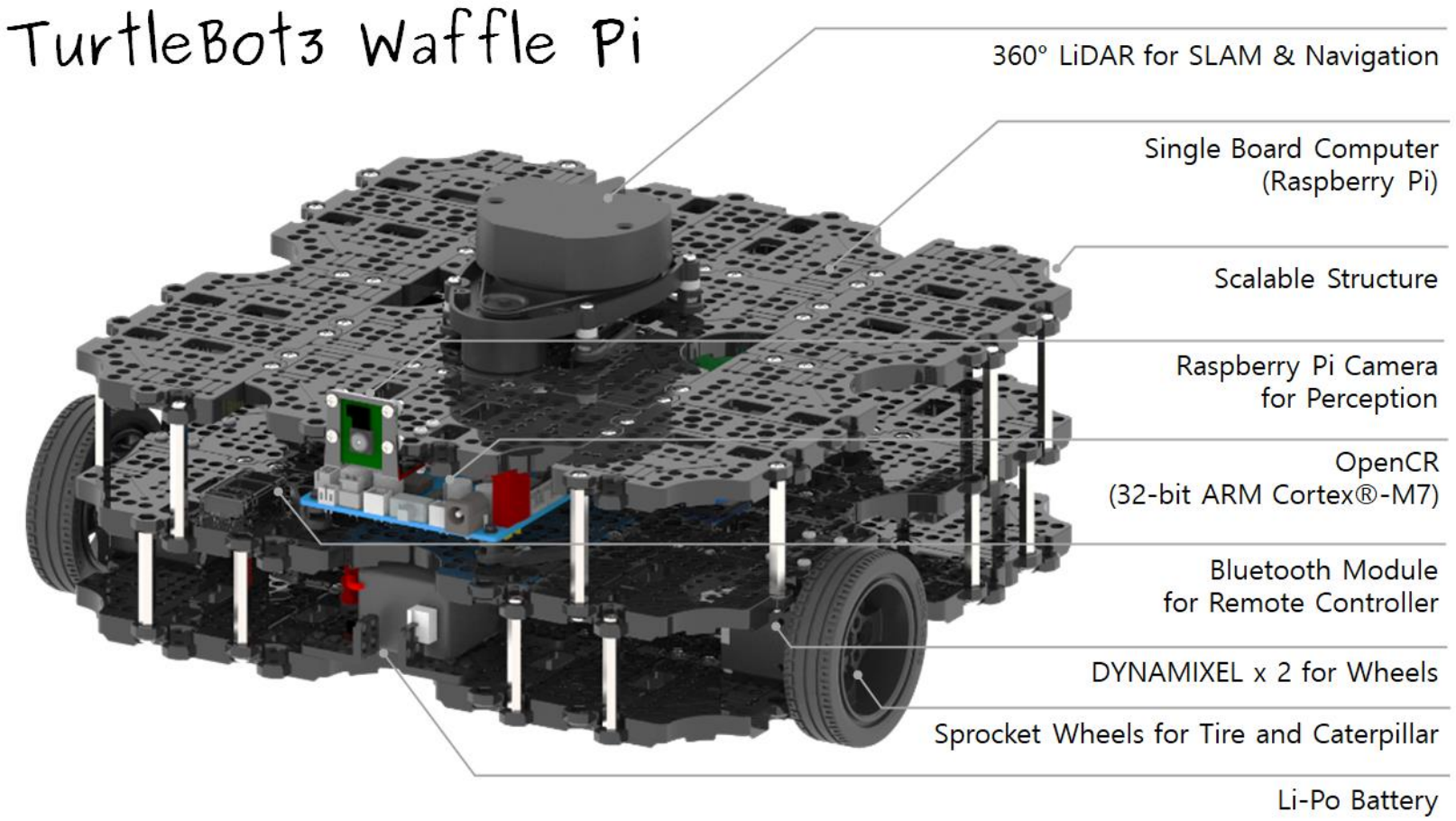
TurtleBot3
Waffle Pi



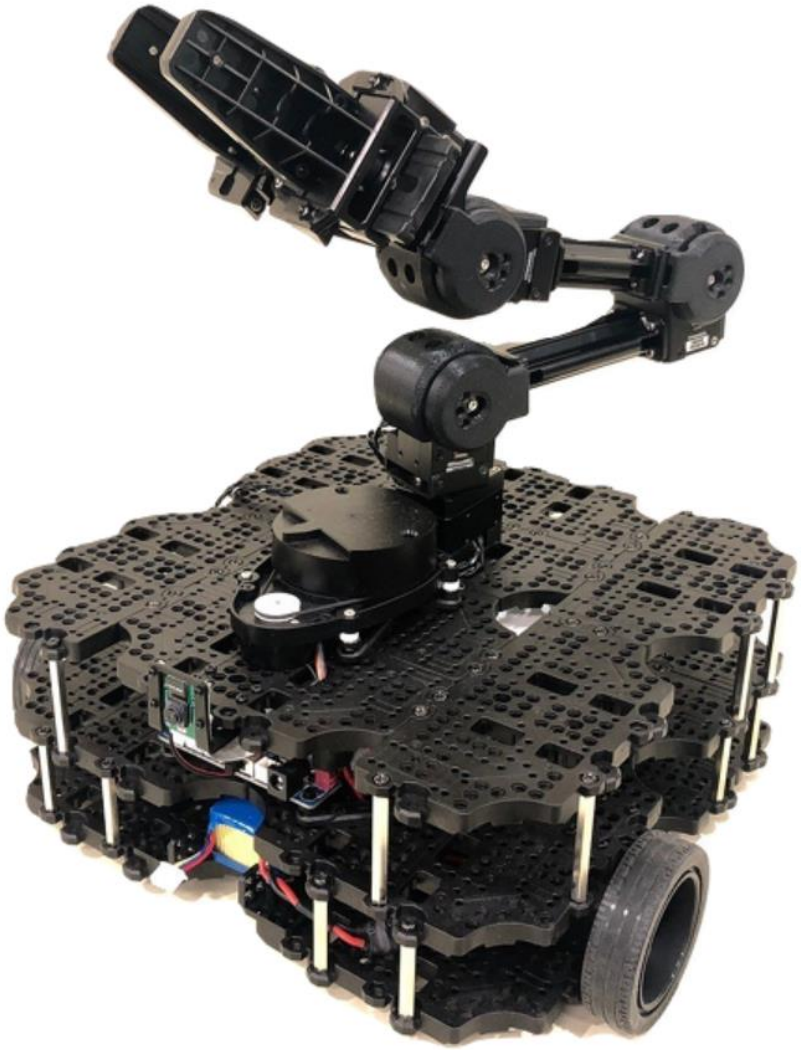
TurtleBot3 Burger



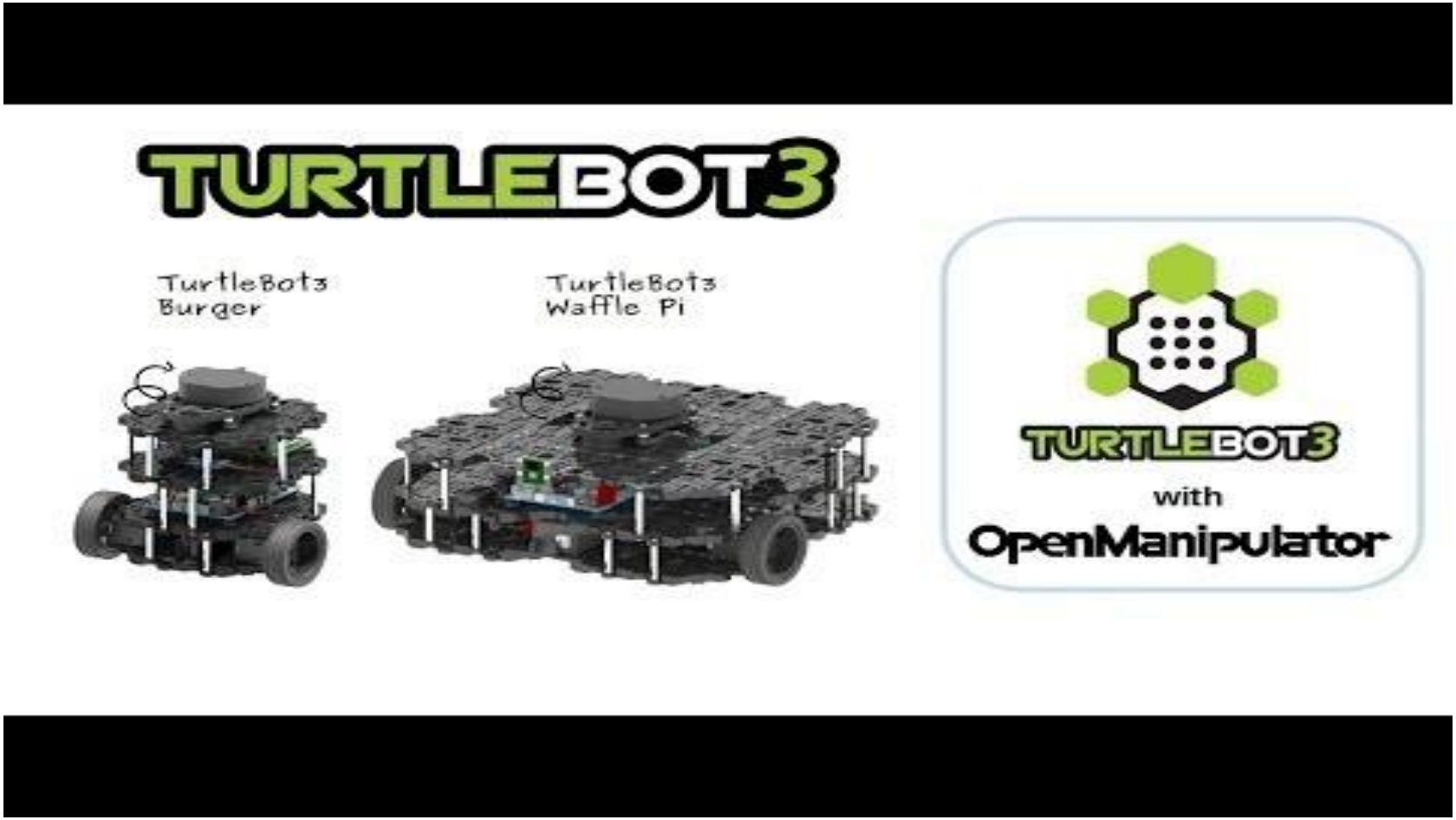
TurtleBot3 Waffle Pi



TurtleBot3 with OpenManipulator-X



TurtleBot3 with OpenManipulator-X

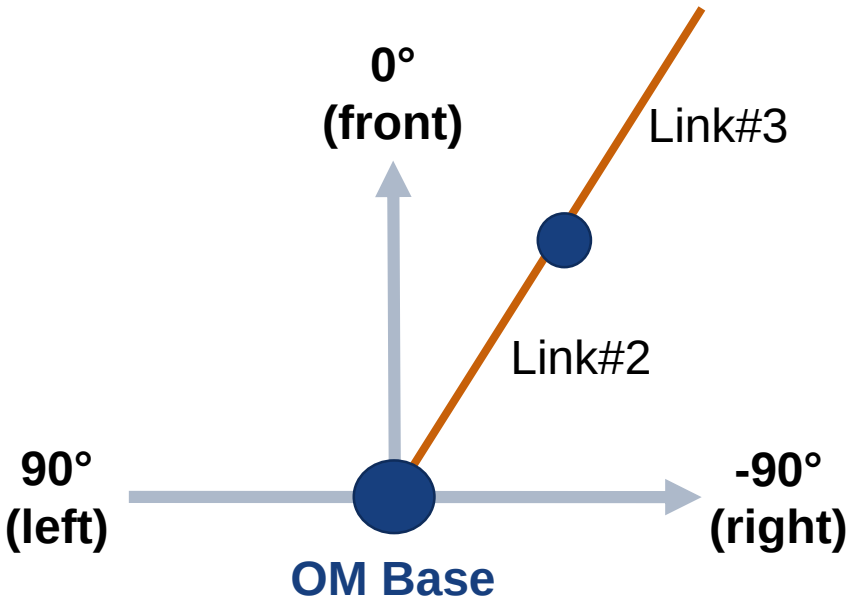


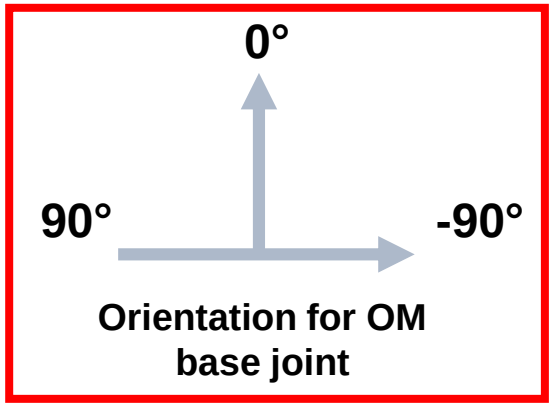
Source: https://youtu.be/P82pZsqpBg0

2. OPEN MANIPULATOR-X HANDS-ON

Orientation for OM
RECAP

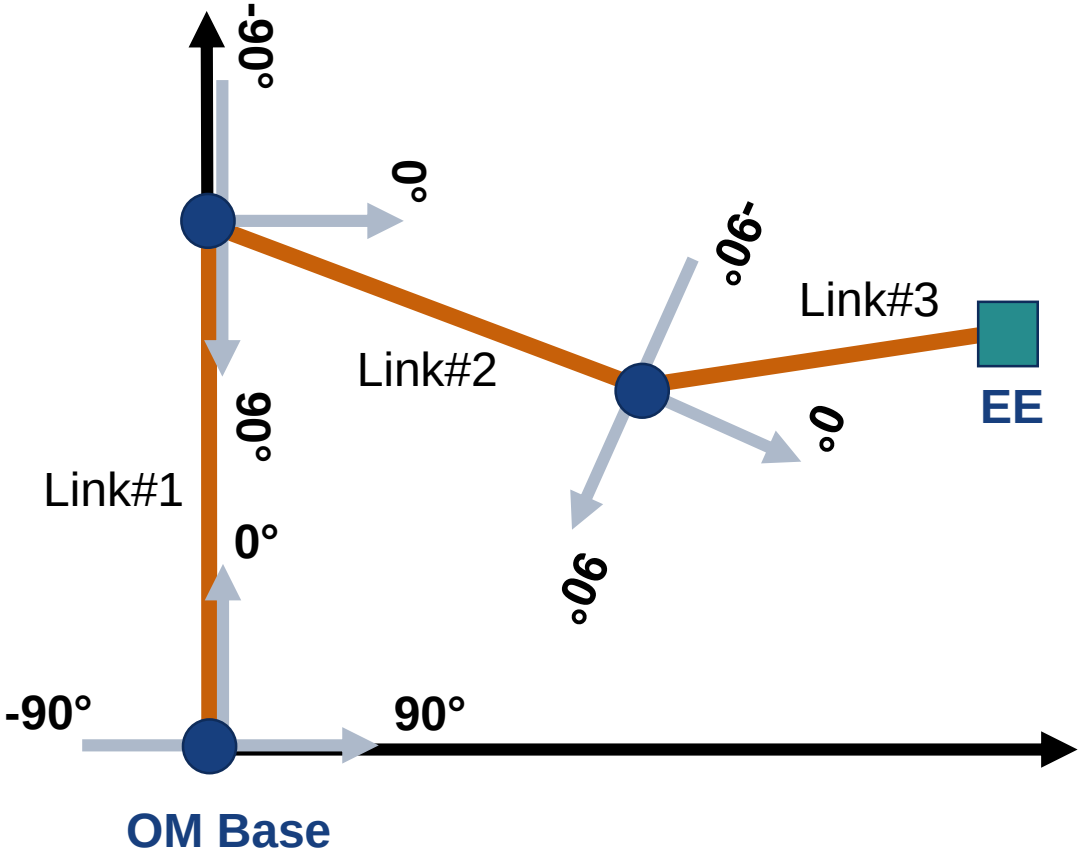
Top View:

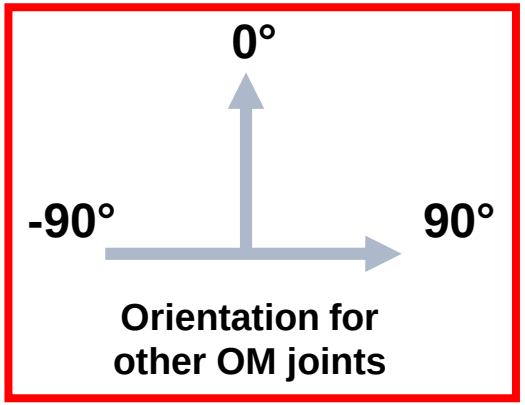




Orientation for OM
RECAP

Side View:





RECAP from Robotics System Course

Steps to control the physical OM:

1. Run `$ roscore` on one terminal
2. Prepare the setup for the OM: Connect the wires, switch on the power, and manually bring up the OM
3. On another terminal, activate the controller; this will lock the motors

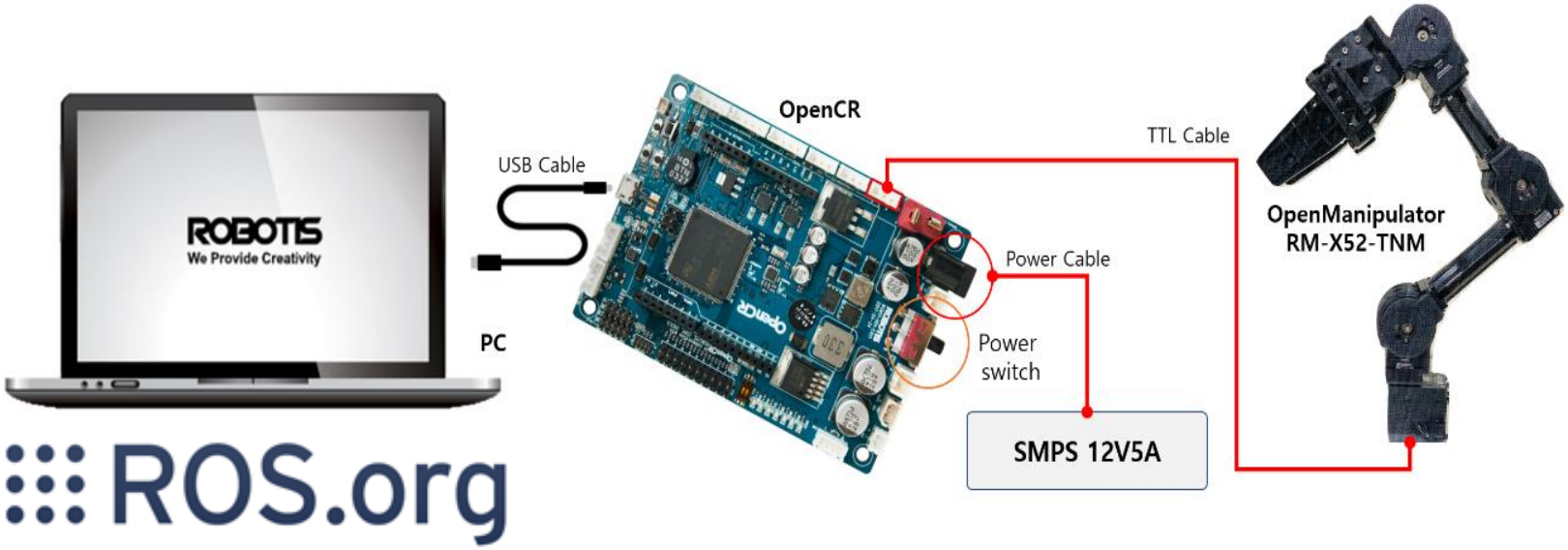
```
$ roslaunch open_manipulator_controller
open_manipulator_controller.launch
```

4. ROSRUN your codes

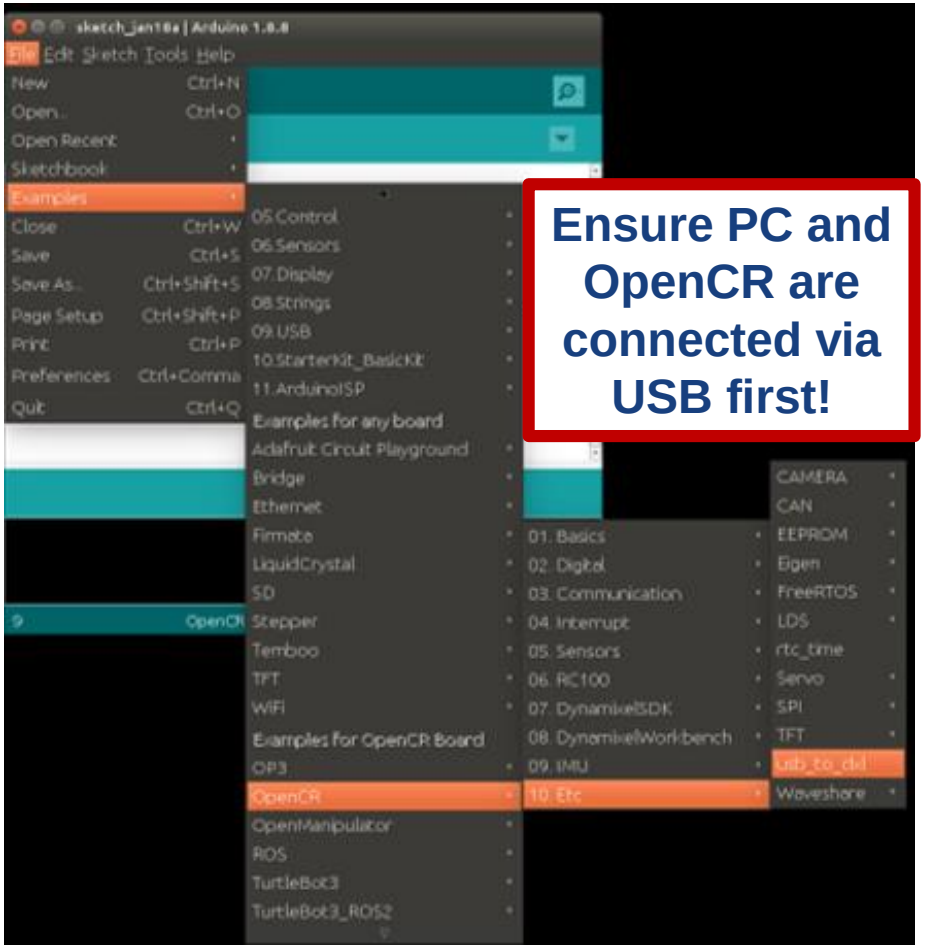
```
$ rosrun robotics control_om.py
```

Using ROS
as the Controller

Setup Configuration:



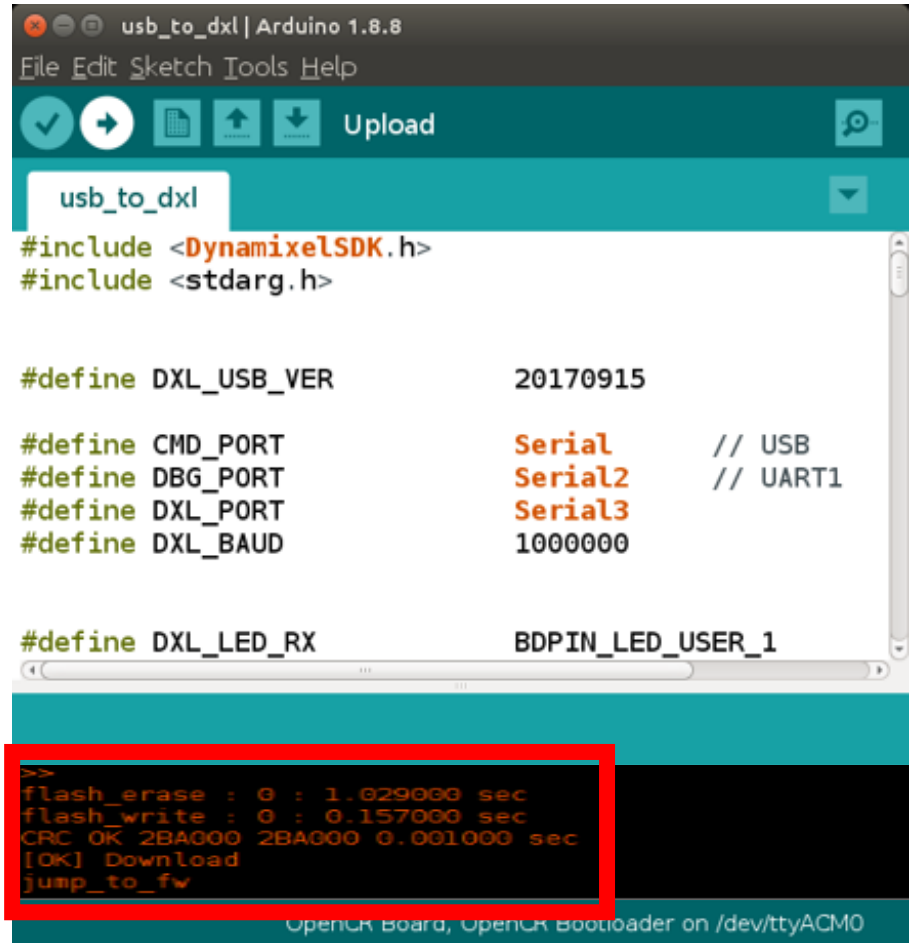
Using ROS:
OpenCR Setup (do it ONCE!):



Ensure PC and OpenCR are connected via USB first!

1. **Open Arduino IDE**
2. **Go to** `Examples` → `OpenCR` → `10.Etc` → `usb_to_dxl` to open the `usb_to_dxl` example source code

Using ROS:
OpenCR Setup (do it ONCE!):



3. Click Upload. If launched successfully, the terminal will appear as follows (left)
   - `jump_to_fw` will appear at the end of the line

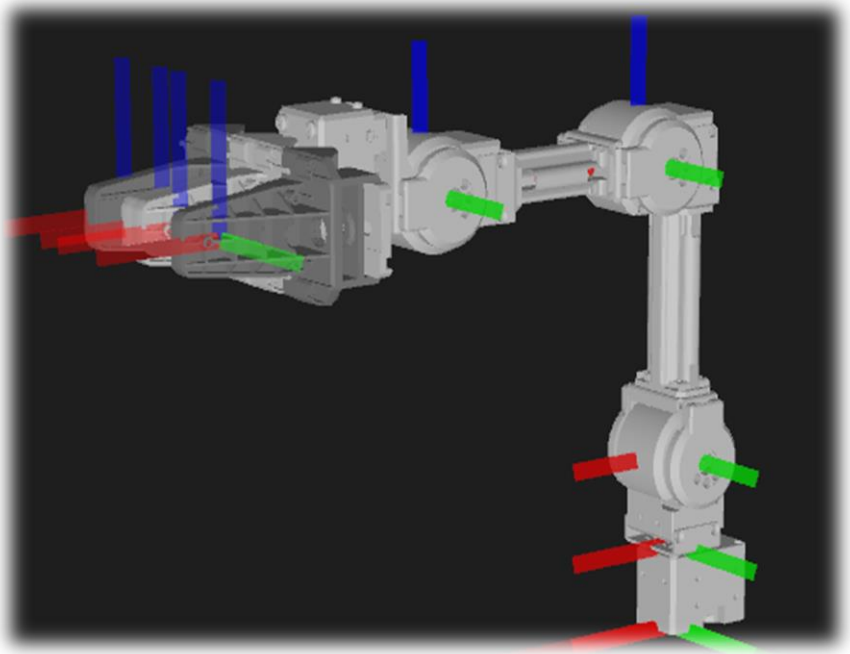
Using ROS: Launch Controller

Important!!!

Before you launch the controller codes, manually lift the arm to a "L" position first!!!

With the end effector (i.e. the gripper) facing the front

(refer to right image)



Using ROS: Launch Controller

Important!!!

Before you close the terminal with the launched controller codes OR before you switch off the power, manually support the arm first!!! And slowly put it down.

If not, the arm motors will be switch off, and it will fall and may injure someone or damage the robot/sensors

Using ROS: Launch Controller

```
SUMMARY
========

PARAMETERS
 * /open_manipulator/control_period: 0.01
 * /open_manipulator/moveit_sample_duration: 0.05
 * /open_manipulator/planning_group_name: arm
 * /open_manipulator/using_moveit: False
 * /open_manipulator/using_platform: True
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES
  /
    open_manipulator (open_manipulator_controller/open_manipulator_controller)

ROS_MASTER_URI=http://localhost:11311

process[open_manipulator-1]: started with pid [23452]
Joint Dynamixel ID : 11, Model Name : XM430-W350
Joint Dynamixel ID : 12, Model Name : XM430-W350
Joint Dynamixel ID : 13, Model Name : XM430-W350
Joint Dynamixel ID : 14, Model Name : XM430-W350
Gripper Dynamixel ID : 15, Model Name :XM430-W350
[ INFO] [1544509070.096942788]: Succeeded to init /open_manipulator
```

*****Ensure the ARM is switched on first**

Type (in New Terminal):

- `$ roscore`
- `$ roslaunch open_manipulator_controller open_manipulator_controller.launch`
- If launched successfully, the terminal will appear as follows (left)
- You are ready to control your robotic arm

Using ROS: Launch Controller

Note that if you can't launch your controller successfully, double check if the USB port in the launch file is defined correctly

- From Home, search and open `open_manipulator_controller.launch`
- Ensure that the following line is as such:

```
<arg name="dynamixel_usb_port"
default="/dev/ttyACM0"/>
```

Inverse Kinematics (IK)

- Remember Project 1 of the Robotics System course?
- Is it practical to calculate the joint values required like what you did?
- Note that Project 1 is still an important practice to enable you to understand the mechanisms behind IK

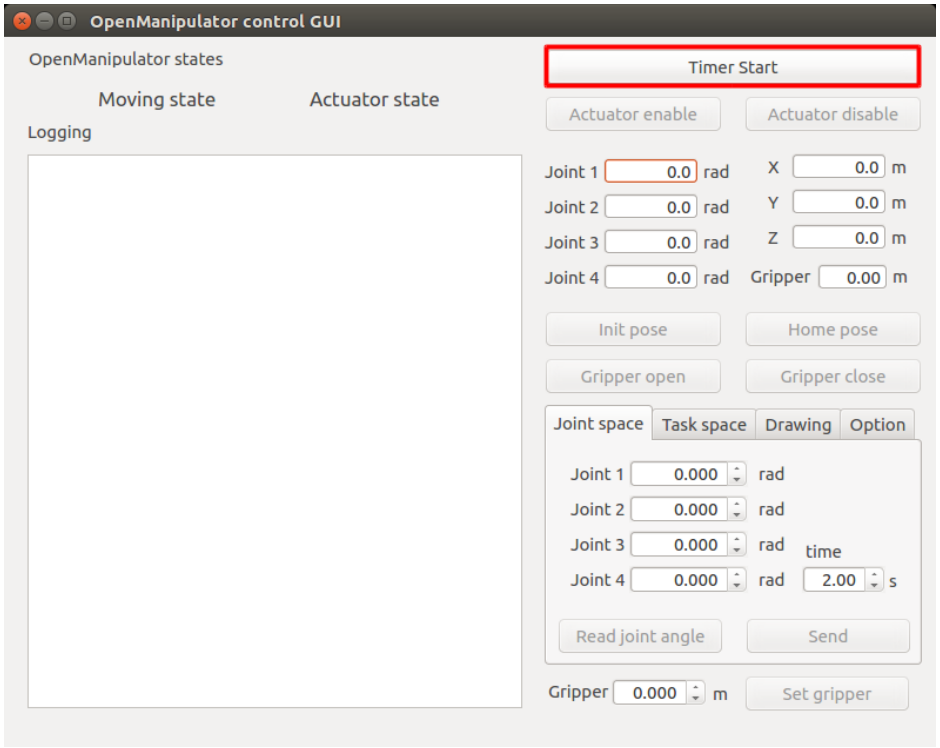
Practical Ways to Solve IK

a) Using OM Control GUI

b) Using rospy (SetKinematicsPose)

c) Hand Guiding

(a) Using OM Control GUI to Solve IK



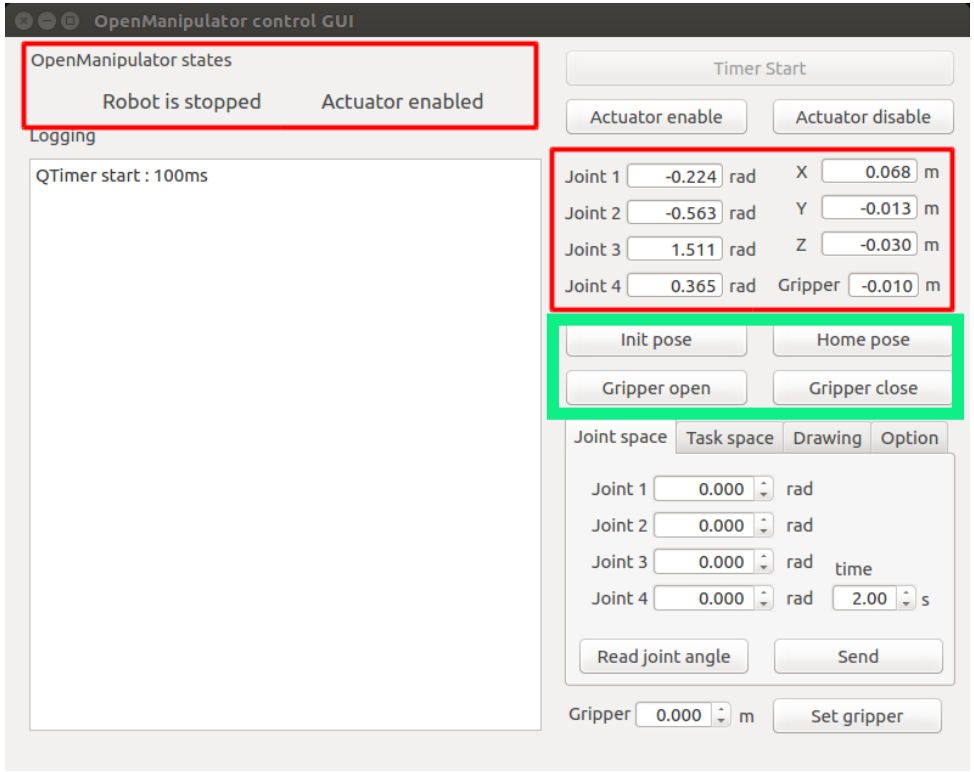
*****Ensure the ARM is switched on and the Controller is activated first**

Type (in New Terminal):

- `$ roslaunch open_manipulator_control_gui open_manipulator_control_gui.launch`
- This will open the GUI for the arm control
- Click “Timer Start” & “Actuator enable” and you can start controlling the arm; observe the arm for response

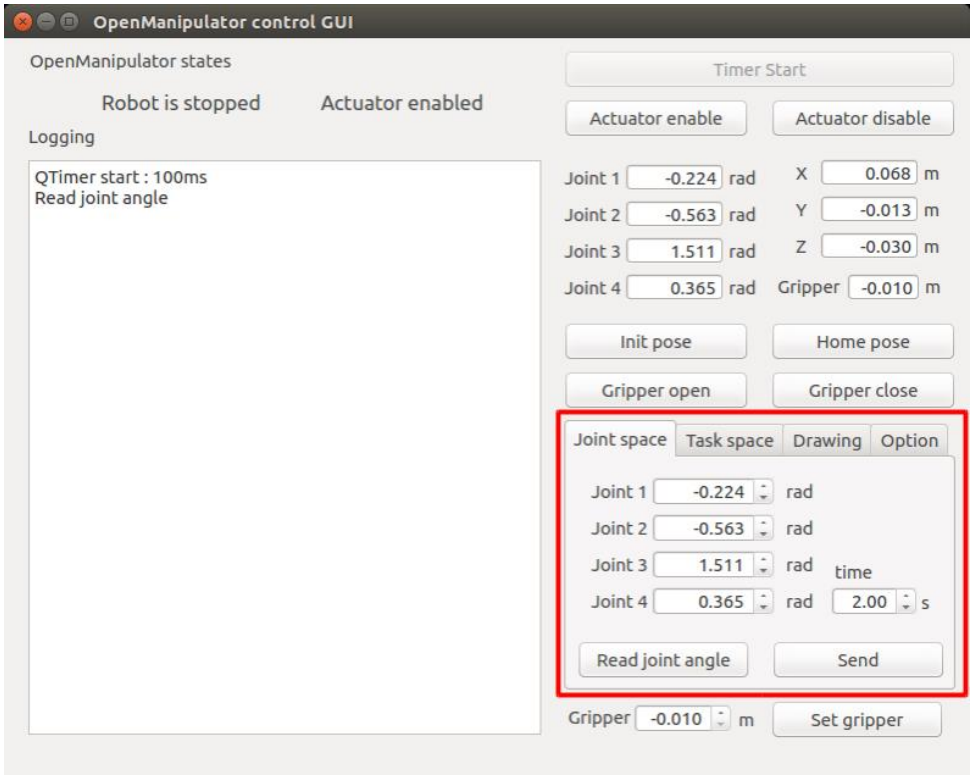
(a) Using OM Control GUI to Solve IK

- You can **check the status of the arm** (i.e. joint states, kinematics pose, end effector position on X-Y-Z)
- To manipulate the arm to get it into **simple poses** (i.e. Init pose, Home pose, gripper open/close), click on the various buttons (in green box) one-by-one



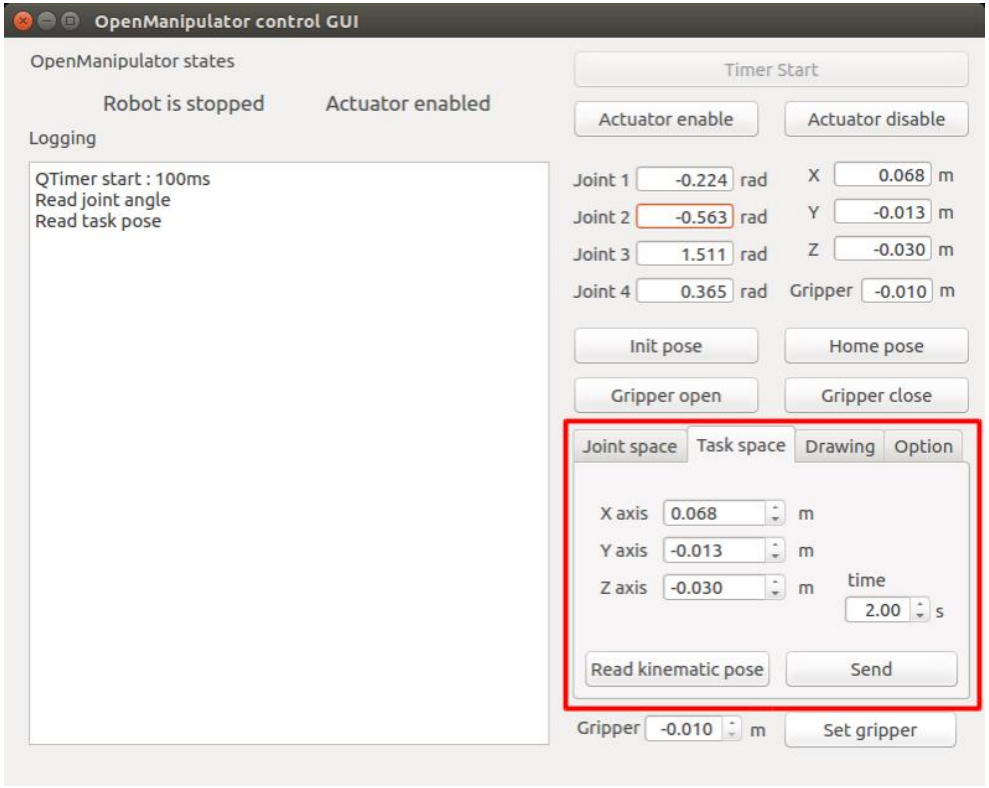
(a) Using OM Control GUI to Solve IK

To manipulate the arm in the joint space (i.e. **Forward Kinematics**), enter the joint angles and trajectory time; click “Send” when done:



(a) Using OM Control GUI to Solve IK

To manipulate the arm in the task space (i.e. **Inverse Kinematics**), enter the X-Y-Z values for your end effector's desired position; click "Send" when done:



(b) Using Python (SetKinematicsPose)

Important!!!

Before you close the terminal with the launched controller codes OR before you switch off the power, manually support the arm first!!! And slowly put it down.

If not, the arm motors will be switch off, and it will fall and may injure someone or damage the robot/sensors

Using ROS:

Control via Python (rospy)

NOTE: These services are messages to operate OpenMANIPULATOR-X or to change the status of DYNAMIXEL of OpenMANIPULATOR.

Service Server List : A list of service servers that open_manipulator_controller has.

- `/open_manipulator/goal_joint_space_path` (open_manipulator_msgs/SetJointPosition)
  The user can use this service to create a trajectory in the joint space. The user inputs the angle of the target joint and the total time of the trajectory.
- `/open_manipulator/goal_joint_space_path_to_kinematics_pose` (open_manipulator_msgs/SetKinematicsPose)
  The user can use this service to create a trajectory in the joint space. The user inputs the kinematics pose of the OpenMANIPULATOR-X end-effector(tool) in the task space and the total time of the trajectory.
- `/open_manipulator/goal_joint_space_path_to_kinematics_position` (open_manipulator_msgs/SetKinematicsPose)
  The user can use this service to create a trajectory in the joint space. The user inputs the kinematics pose(position only) of the OpenMANIPULATOR-X end-effector(tool) in the task space and the total time of the trajectory.
- `/open_manipulator/goal_joint_space_path_to_kinematics_orientation` (open_manipulator_msgs/SetKinematicsPose)
  The user can use this service to create a trajectory in the joint space. The user inputs the kinematics pose(orientation only) of the OpenMANIPULATOR-X end-effector(tool) in the task space and the total time of the trajectory.
- `/open_manipulator/goal_task_space_path` (open_manipulator_msgs/SetKinematicsPose)
  The user can use this service to create a trajectory in the task space. The user inputs the kinematics pose of the OpenMANIPULATOR-X end-effector(tool) in the task space and the total time of the trajectory.

(b) Using Python (SetKinematicsPose)

- Open terminal and type:

```
$ rosrun autonomous setkinematicspose.py
```

- The OM will move to a point determined by the X-Y-Z positions. To alter these positions, open the python file, find the following code section and change the positions in it; these angles are in m:

```
kinematics_pose = KinematicsPose()
kinematics_pose.pose.position.x = 0.0128
kinematics_pose.pose.position.y = 0.2748
kinematics_pose.pose.position.z = 0.1938
```

(b) Using Python (SetKinematicsPose)

setkinematicspose.py

```
#!/usr/bin/env python
# works for actual OM ONLY!

import rospy                                    #import the python library for ROS
from open_manipulator_msgs.msg import JointPosition     #import JointPosition message from the open_manipulator_msgs package
from open_manipulator_msgs.srv import SetJointPosition
from open_manipulator_msgs.msg import KinematicsPose
from open_manipulator_msgs.srv import SetKinematicsPose
from geometry_msgs.msg import Pose
import math

def talker():
    rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
    set_kinematics_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path_to_kinematics_position', SetKinematicsPose)
    set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

    #while not rospy.is_shutdown():
    kinematics_pose = KinematicsPose()
    kinematics_pose.pose.position.x = 0.287
    kinematics_pose.pose.position.y = 0.0
    kinematics_pose.pose.position.z = 0.194
    resp1 = set_kinematics_position('planning_group', 'gripper', kinematics_pose, 3)

    gripper_position = JointPosition()
    gripper_position.joint_name = ['gripper']
    gripper_position.position =  [0.01]     # -0.01 for fully close and 0.01 for fully open
    respg2 = set_gripper_position('planning_group',gripper_position, 3)

    rospy.sleep(3)
    kinematics_pose = KinematicsPose()
    kinematics_pose.pose.position.x = 0.0
    kinematics_pose.pose.position.y = 0.276
    kinematics_pose.pose.position.z = 0.192
    resp1 = set_kinematics_position('planning_group', 'gripper', kinematics_pose, 3)

if __name__== '__main__':
        try:
                talker()
        except rospy.ROSInterruptException:
                pass
```

Note that if you want to use *SetKinematicsPose*, you have to be careful of the coordinates that you select; some of them cannot be solved by the IK solver

WHY so?

(b) Using Python (SetKinematicsPose)

- Open terminal and type:

```
$ rosrun autonomous
setkinematicspose_withorientation.py
```

- The OM will move to a point determined by the X-Y-Z positions and its orientations. To alter these positions/orientations, open the python file, find the following code section and change the positions/orientations in it:

```
kinematics_pose = KinematicsPose()
kinematics_pose.pose.position.x = 0.0128
kinematics_pose.pose.position.y = 0.2748
kinematics_pose.pose.position.z = 0.1938

kinematics_pose.pose.orientation.w = 0.707
kinematics_pose.pose.orientation.x = -0.016
kinematics_pose.pose.orientation.y = 0.016
kinematics_pose.pose.orientation.z = 0.7058
```

(b) Using Python (SetKinematicsPose)

```
#!/usr/bin/env python
# works for actual OM ONLY!

import rospy                                    #import the python library for ROS
from open_manipulator_msgs.msg import JointPosition     #import JointPosition message from the open_manipulator_msgs
package
from open_manipulator_msgs.srv import SetJointPosition
from open_manipulator_msgs.msg import KinematicsPose
from open_manipulator_msgs.srv import SetKinematicsPose
from geometry_msgs.msg import Pose
import math

def talker():
        rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
        set_kinematics_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path_to_kinematics_pose',
SetKinematicsPose)
        set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

        #while not rospy.is_shutdown():
        kinematics_pose = KinematicsPose()
        kinematics_pose.pose.position.x = 0.2867
        kinematics_pose.pose.position.y = 0.0
        kinematics_pose.pose.position.z = 0.194

        kinematics_pose.pose.orientation.w = 0.999
        kinematics_pose.pose.orientation.x = 1.765
        kinematics_pose.pose.orientation.y = 0.023
        kinematics_pose.pose.orientation.z = 0.0

        resp1 = set_kinematics_position('planning_group', 'gripper', kinematics_pose, 3)

        gripper_position = JointPosition()
        gripper_position.joint_name = ['gripper']
        gripper_position.position =  [0.01]     # -0.01 for fully close and 0.01 for fully open
        respg2 = set_gripper_position('planning_group',gripper_position, 3)

        rospy.sleep(3)
        kinematics_pose = KinematicsPose()
        kinematics_pose.pose.position.x = 0.0128
        kinematics_pose.pose.position.y = 0.2748
        kinematics_pose.pose.position.z = 0.1938

        kinematics_pose.pose.orientation.w = 0.707
        kinematics_pose.pose.orientation.x = -0.016
        kinematics_pose.pose.orientation.y = 0.016
        kinematics_pose.pose.orientation.z = 0.7058

        resp1 = set_kinematics_position('planning_group', 'gripper', kinematics_pose, 3)
```

(b) Using Python (SetKinematicsPose)

Important!!!

Before you close the terminal with the launched controller codes OR before you switch off the power, manually support the arm first!!! And slowly put it down.

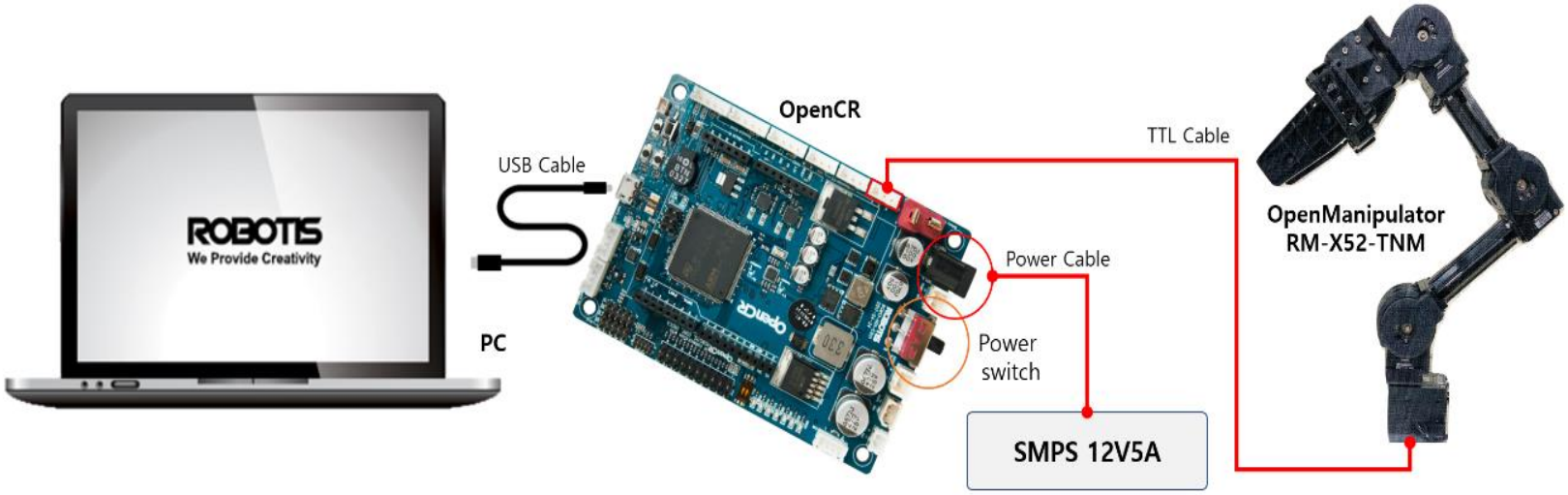
If not, the arm motors will be switch off, and it will fall and may injure someone or damage the robot/sensors

(c) Hand Guiding to solve IK

Source: https://www.youtube.com/watch?v=9mE9QgAro8A

Using OpenCR
as the Controller

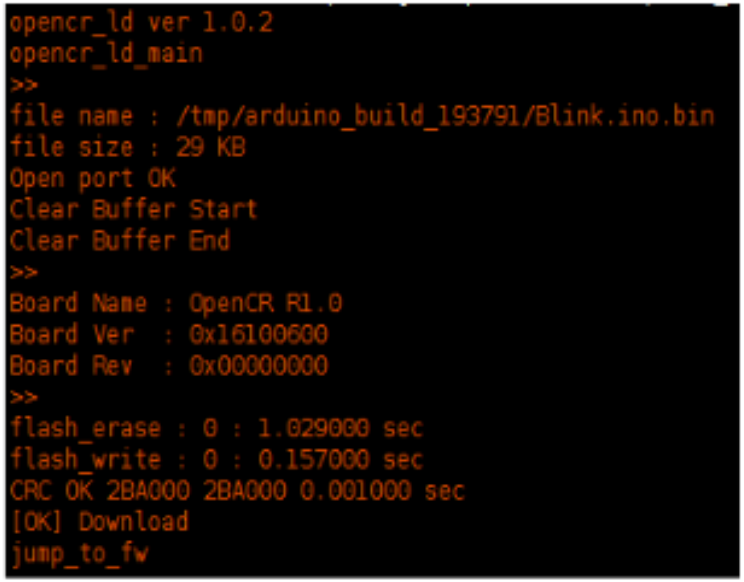
Setup Configuration (same setup):



Using OpenCR: Operation

Upload Controller Program to OpenCR:

1. **Open Arduino**
2. **Go to *File* → *Examples* → *OpenManipulator* → *example* → *Chain* → *open_manipulator_chain***
3. **Upload the example source to OpenCR;** upload of firmware is successful if `jump_to_fw` appears at the end of the line (refer to right image)



Using OpenCR: Operation

Launch Processing (GUI):

1. **First remove the USB cable, Connect the power supply to OpenCR (do not switch on power yet)**
2. **Manually lift the OM to a "L" position**
3. **Hold the OM in that position and switch on the OpenCR switch**
4. Check if the motors of OM are torque enabled (i.e. the motors are locked in and prevent you from moving the OM); you may let go the OM once the motors are torque enabled
5. **Connect the OpenCR and PC with a USB cable**

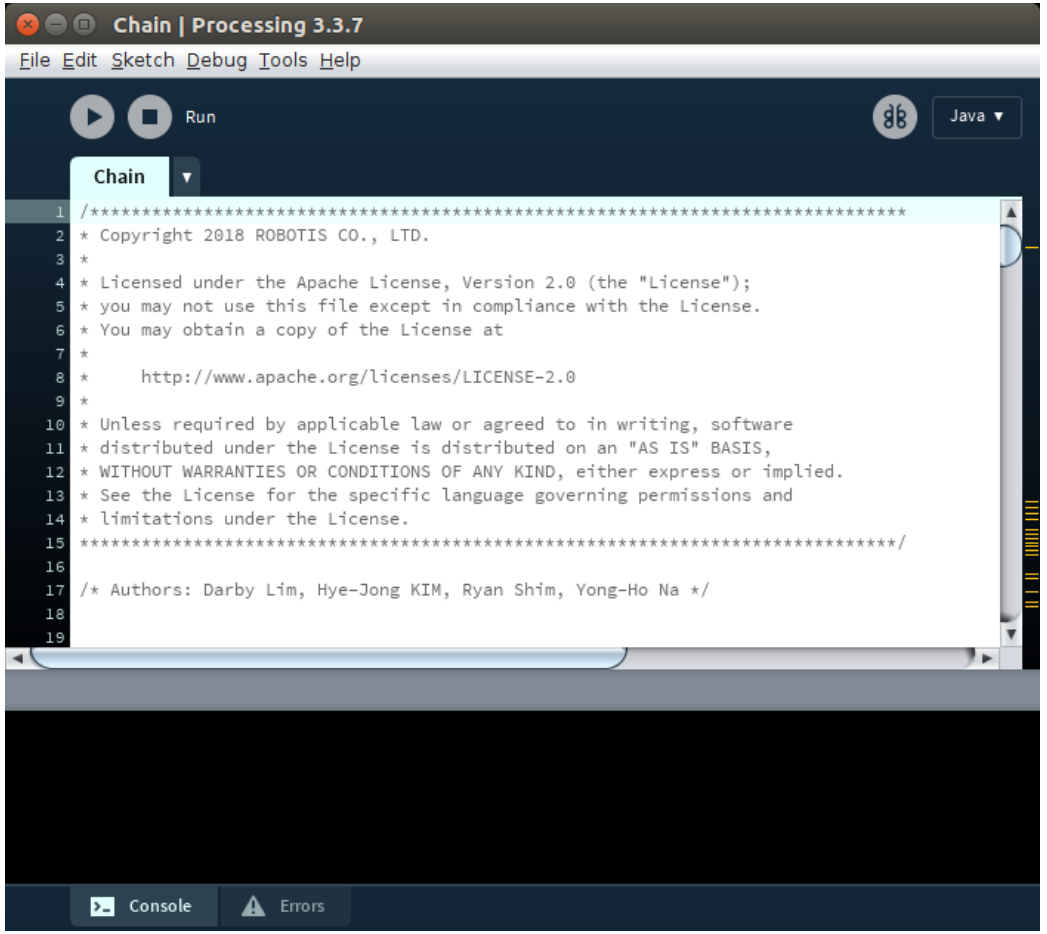
Using OpenCR: Operation

Launch Processing (GUI):

6. Open the Processing application
7. Click *File* → *Open*
8. Search downloaded processing file from ***~/catkin_ws/src/*** → *open_manipulator_processing* → *Chain* → *Chain.pde* and **open it on Processing IDE**

Using OpenCR: Operation

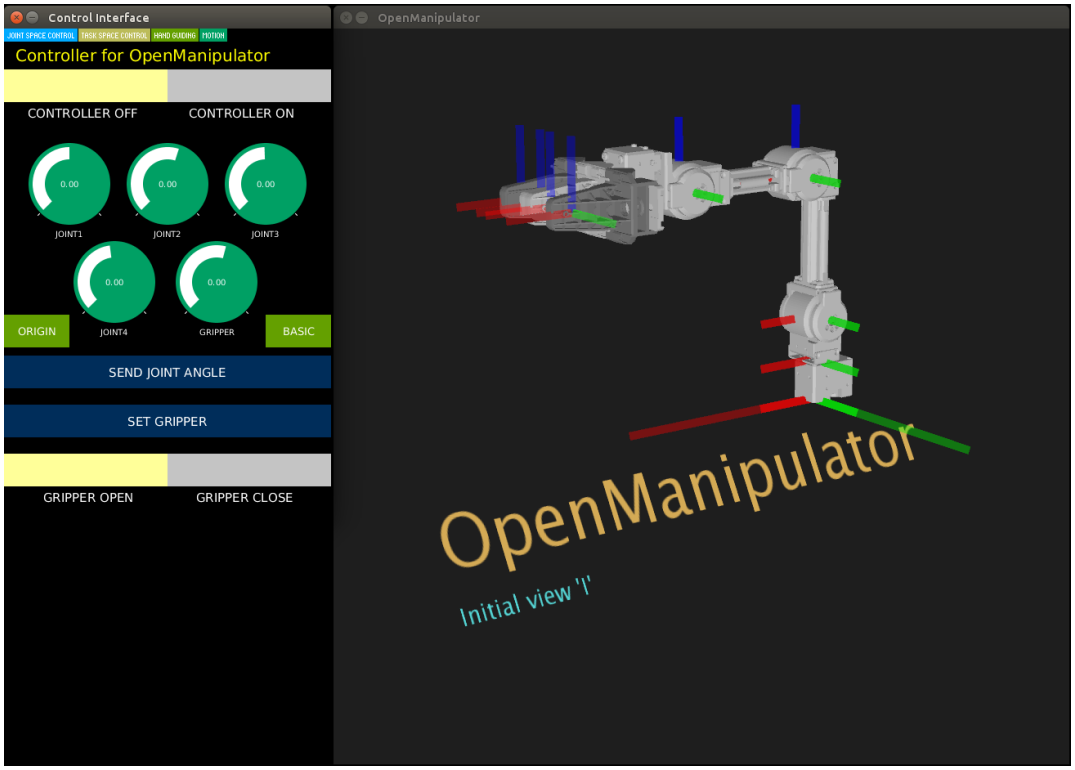
Example of Chain code file in the Processing application:



Using OpenCR: Operation

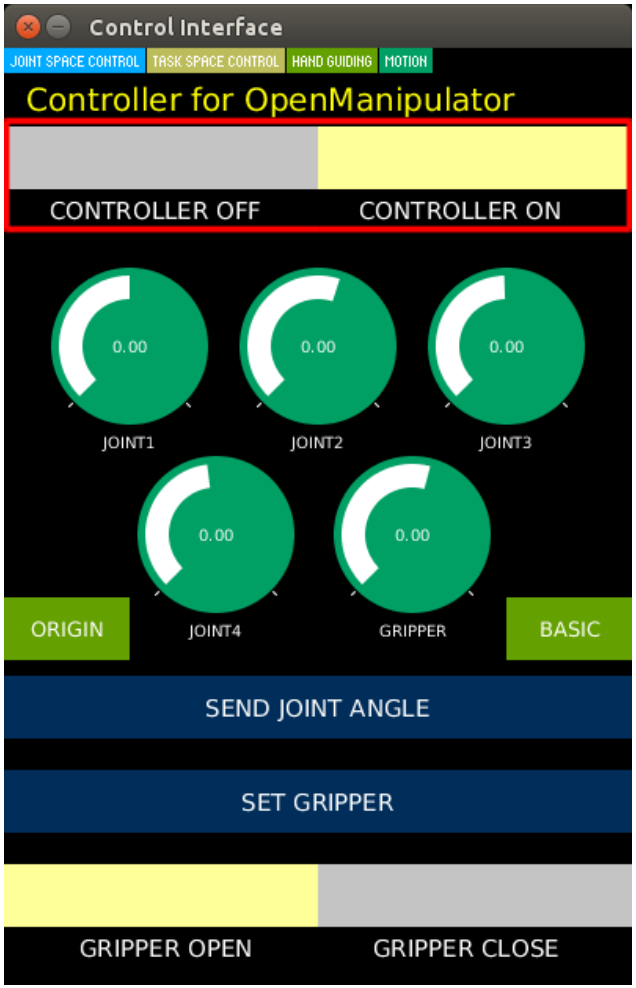
Launch Processing (GUI):

9. Run the processing source code, and the following graphical GUI will be displayed



Using OpenCR: Operation

Using Control Interface to control arm:



To start manipulating the arm, click the virtual toggle button to CONTROLLER ON

Using OpenCR: Operation

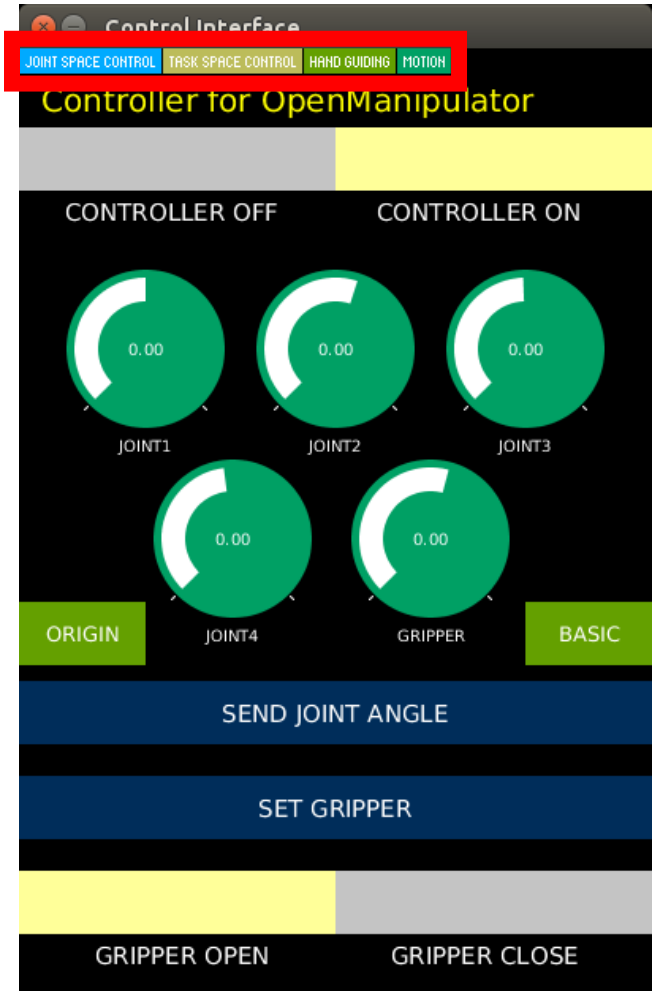
Important!!!

If you want to set the CONTROLLER to OFF, manually support the arm first!!! And slowly put it down.

If not, the arm motors will be switch off, and it will fall and may injure someone or damage the robot/sensors

Using OpenCR: Operation

Using Control Interface to control arm:

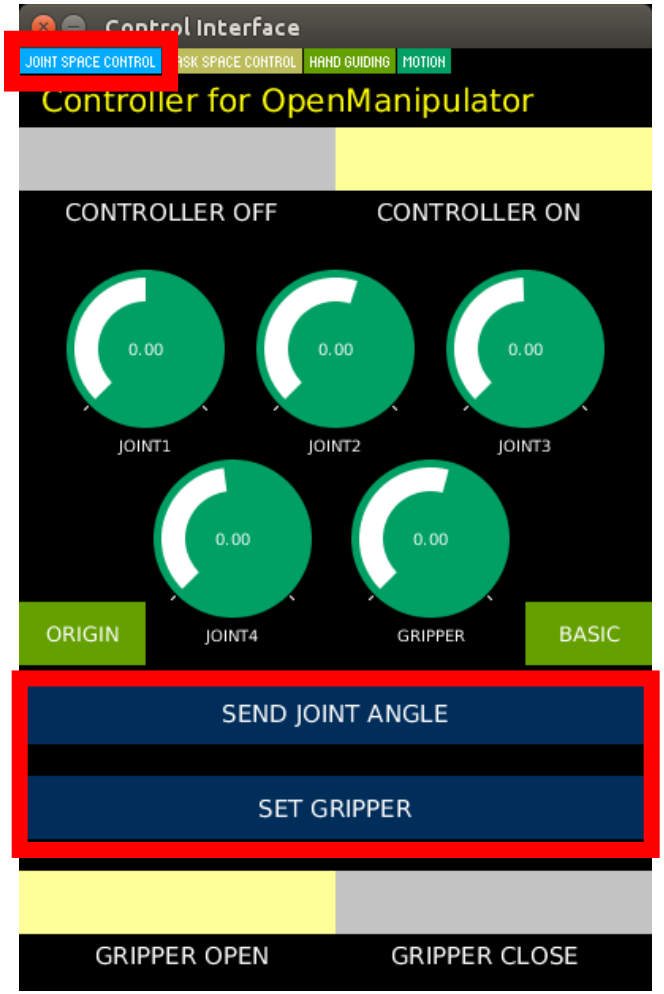


3 Main Modes of Control:

1. **Joint space**
2. **Task space**
3. **Hand guiding**

Using OpenCR: Operation

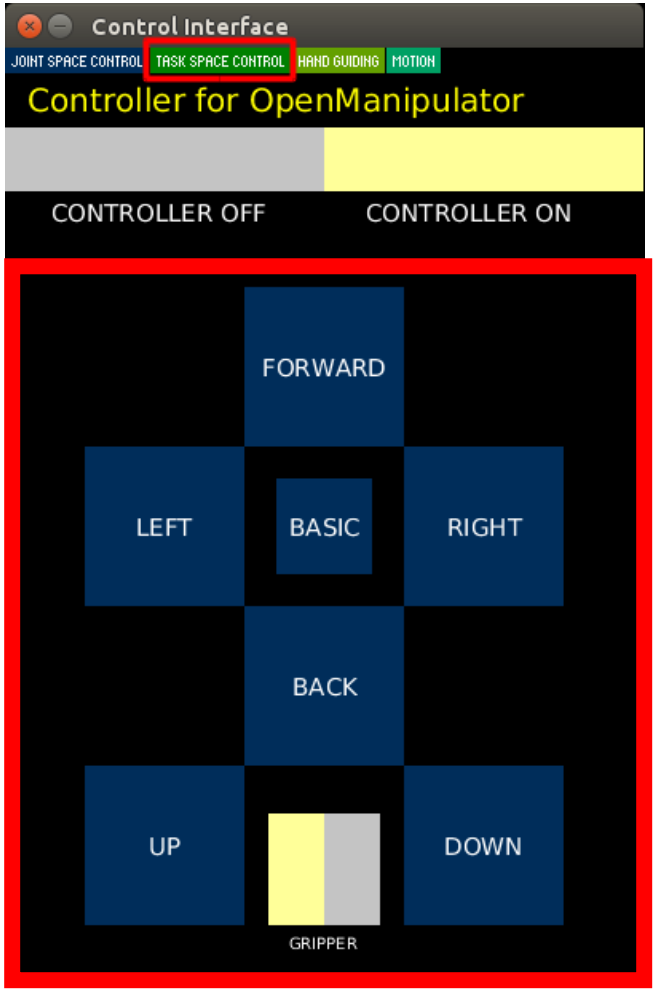
(A) Joint space control



- Set the joint angles and click the SEND JOINT ANGLE button
- Set the gripper parameter and click the SET GRIPPER button

Using OpenCR: Operation

(B) Task space control



- **Click the desired direction buttons to manipulate the ARM**

Using OpenCR: Operation

(C) Hand Guiding

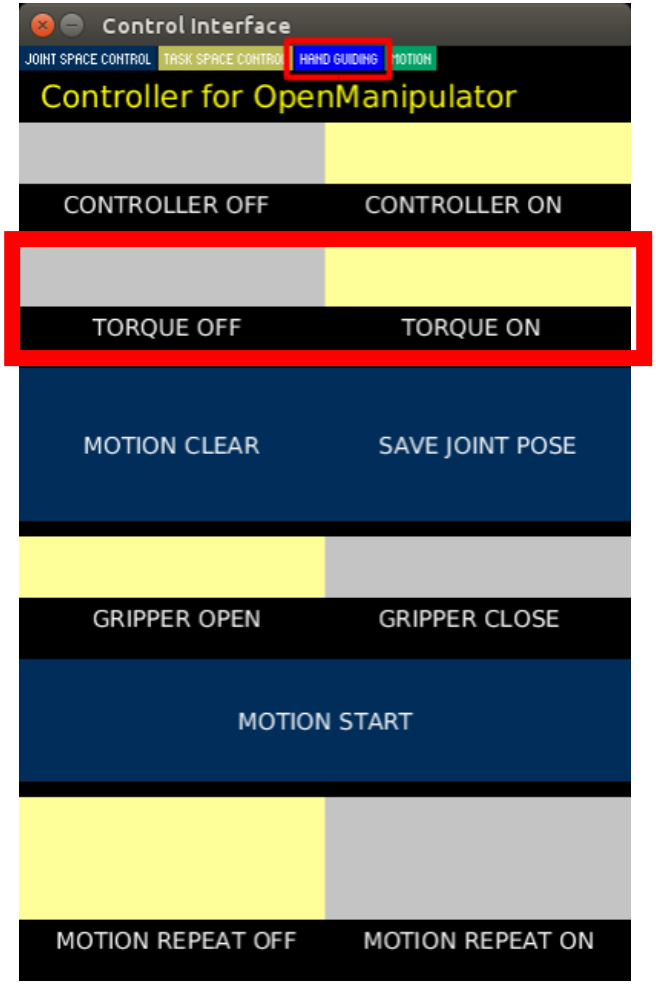
Important!!!

Before you switch off the torque during the hand guiding session, manually support the arm first!!! And you can start follow the instructions in the following slides.

If not, the arm motors will be switch off, and it will fall and may injure someone or damage the robot/sensors

Using OpenCR: Operation

(C) Hand Guiding

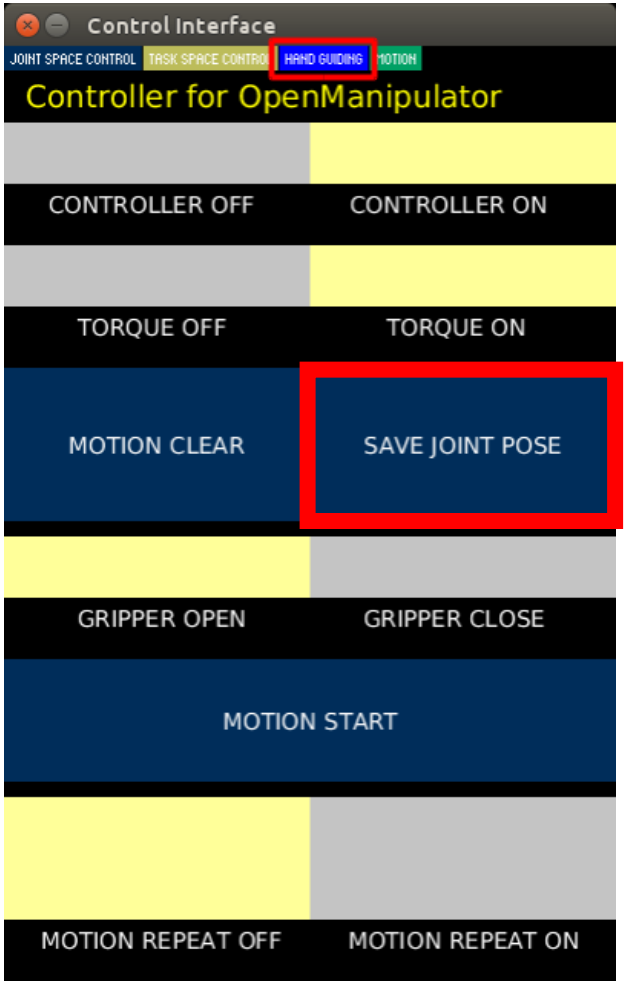


Steps:

1. Hold the ARM by your hand (to prevent it from falling) and click the toggle button to TORQUE OFF
2. Move the ARM to the desired pose using your hand

Using OpenCR: Operation

(C) Hand Guiding

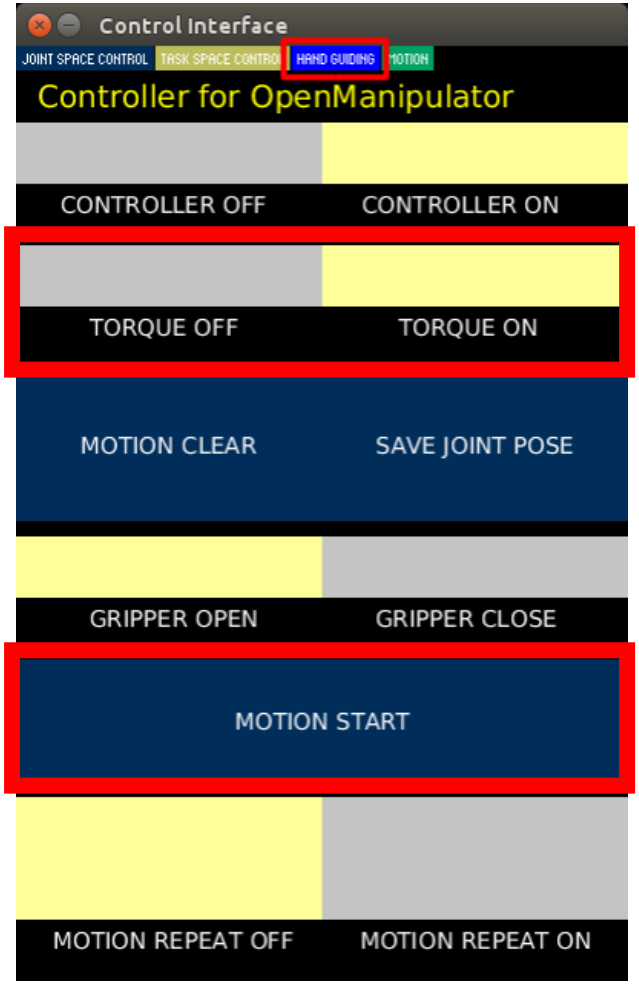


Steps:

3. Click the SAVE JOINT POSE to save the present pose
4. Repeat step 2 & 3 to create the demonstration

Using OpenCR: Operation

(C) Hand Guiding

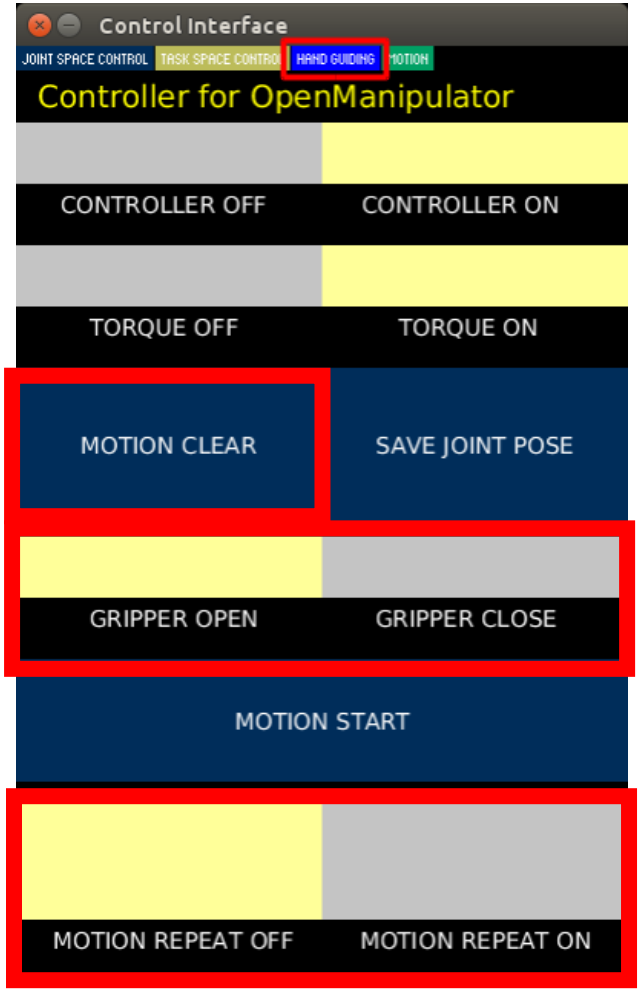


Steps:

5. Once done, click the toggle button to TORQUE ON
6. Click the MOTION START button to start the saved poses

Using OpenCR: Operation

(C) Hand Guiding



Other considerations:

- Use the GRIPPER OPEN/CLOSE button to control the gripper
- Click MOTION CLEAR if you want to delete memory of the old motion paths in order to create new ones
- Click MOTION REPEAT ON if you want to repeat the demonstration

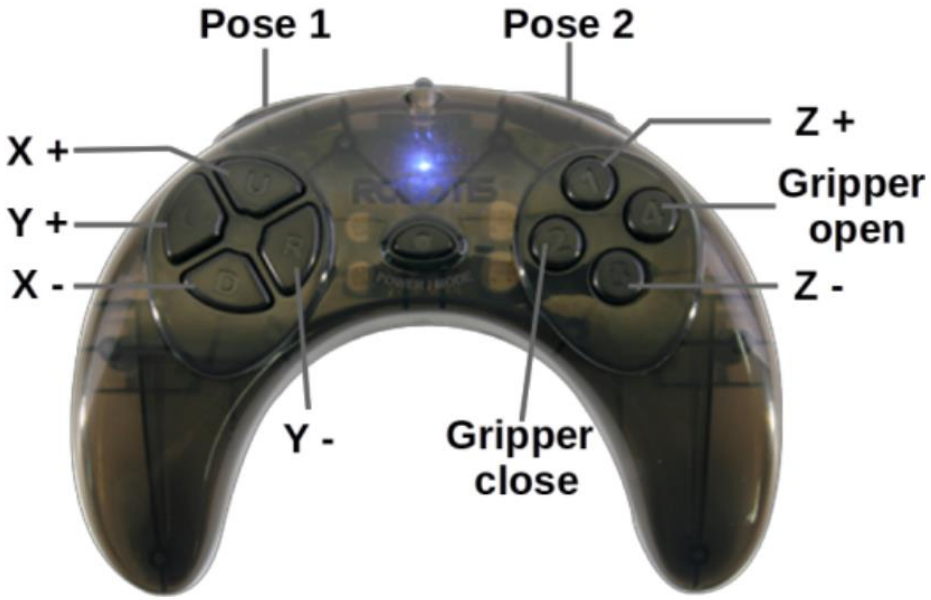
Using OpenCR: Operation

Important!!! Do the following Steps before you start using the RC-100 Controller:

1. **Remove USB connection to the PC;** you will not be able to use RC-100 to control the OM if the OpenCR and PC are connected via the USB
2. **Connect the power supply to OpenCR (do not switch on power yet)**
3. **Manually lift the OM to a "L" position with the gripper facing the front**
4. **Hold the OM in that position and switch on the OpenCR power switch**
5. Check if the motors of OM are torque enabled; you can let go the OM once the motors are torque enabled
6. **Switch on the RC-100 Controller** and you can start using it to control the OM

Using OpenCR: Operation

Using RC-100 Controller (Bluetooth):



The settings for this controller is included in the OpenCR firmware for the ARM

Group Project 2

There are various types of autonomous robots out there in the world. One of the popular type involves the Hand-Guiding method, which rapidly and intuitively program a robotic arm by teaching it the desired trajectories (and hence the respective joint angles) it has to execute in order to achieve its goal(s). One popular application of this method is at manufacturing lines whereby repetitive tasks are common and frequent reprogramming activities are required.

For this project, we will **compare the pros/cons among:**

a) **Manual-based robots (using the RC100 controller),**

b) **Programmable autonomous robots (Python-controlled method), and**

c) **Adaptive autonomous robots (precise Hand-Guiding teaching method)**

Group Project 2

In groups, you are to:

1. **Achieve 2 main goals**
   a) Transfer the BIG item from Position A to B
   b) Repeat this task 3 times (you can reuse the item and put it back to Position A each time each task is completed)
   c) Change Positions A and B to random new positions, and do (a) and (b) again
   d) Take the SMALL item and do (a) and (b) again; use the same positions from (c)
2. Firstly, try using the **RC100 controller** to remotely control the OM to achieve the goals. Reflect on the pros/cons of this method
3. Secondly, write a **Python code(s) (i.e. rospy)** to control the OM to achieve the goals (rely on FK/IK techniques as taught in the RBS WS and previous slides in M3). You may first use the GUI tool as a guide to obtain the required joint angles via inverse kinematics. Reflect on the pros/cons of this method
4. Thirdly, using the **Hand-Guiding teaching method** that we practiced previously, **configure an adaptive autonomous robot** that will automatically achieve the goals. Reflect on the pros/cons of this method

Group Project 2

Note:

- **For Manual Control**, use the current **OpenCR configuration**. Use the RC100 controller as explained in the previous slides
- **For Hand Guiding**, use the current **OpenCR configuration**. Refer to the explanations in the previous slides and/or the README instructions
- **For FK/IK Techniques**, use the **ROS configuration**. Remember to reconfigure and upload the correct firmware using Arduino. You can use the given `Project2.py` file as a reference template for this part

Group Project 2

RECAP on procedures to rosrun:

1. Run `$ roscore` on one terminal
2. Prepare the setup for the OM: Connect the wires, switch on the power, and manually bring up the OM
3. On another terminal, activate the controller; this will lock the motors

```
$ roslaunch open_manipulator_controller
open_manipulator_controller.launch
```

4. On another terminal, ROSRUN your codes

```
$ rosrun autonomous Project2.py
```

Group Project 2

Fill in the comparison table below:

	Manual-based robots (using RC100 controller)	Programmable autonomous robots (using Python codes)	Adaptive autonomous robots (using hand-guiding)
Pros			
Cons			

Group Project 2

Improving autonomous level of system:

- Upgrade the hand guiding system into an intelligent autonomous robot by **suggesting a newly improved robotic arm system with higher autonomy by drafting out an architecture design** that represents this system
- Assume that:
  - There is no budget limitation for your system
  - The base equipment (i.e. the OM itself, openCR board and PC) remains the same. You are allowed to suggest additional devices/technologies to improve the system

Group Project 2

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

1. **The filled comparison table (Word/PDF formats are all accepted)**
2. **Python codes (.py)** to achieve goals for FK/IK techniques
3. **Architecture design** of an improved robotic arm system with a higher autonomous level

Please upload all your works in Luminus with your names/student IDs as part of the title:

(e.g. DANIEL_KEN_TAN_Group_Project_2)

OR

(e.g. AXX_AXX_AXX_Group_Project_2)

End of Module 3

THANK YOU for your kind attention!

Main References

- Self-Driving Cars: A Survey, Cornell University, by Badue, et al.
- Autonomous Systems – An Architectural Characterization, Cornell University, by Joseph Sifakis
- Human and robot behavior modeling for probabilistic cognition of an autonomous service robot, RO-MAN 2008, by Schmidt-Rohr, et al.
- Safe, Autonomous and Intelligent Vehicles, Unmanned System Technologies, Springer, by Yu, et al.
- Risk Analysis of Autonomous Vehicles in Mixed Traffic Streams, Rowan University, by Bhavsar, et al.
- Ethical and Social Aspects of Self-Driving Cars, Cornell University, by Holstein, et al.
- https://www.itransition.com/blog/three-types-of-autonomous-vehicle-sensors-in-self-driving-cars
- https://www.elprocus.com/different-types-of-autonomous-robots-and-real-time-applications/
- https://waypointrobotics.com/blog/what-autonomous-robots/
- http://wiki.ros.org/ROS/Introduction
- http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#turtlebot
- http://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/#overview