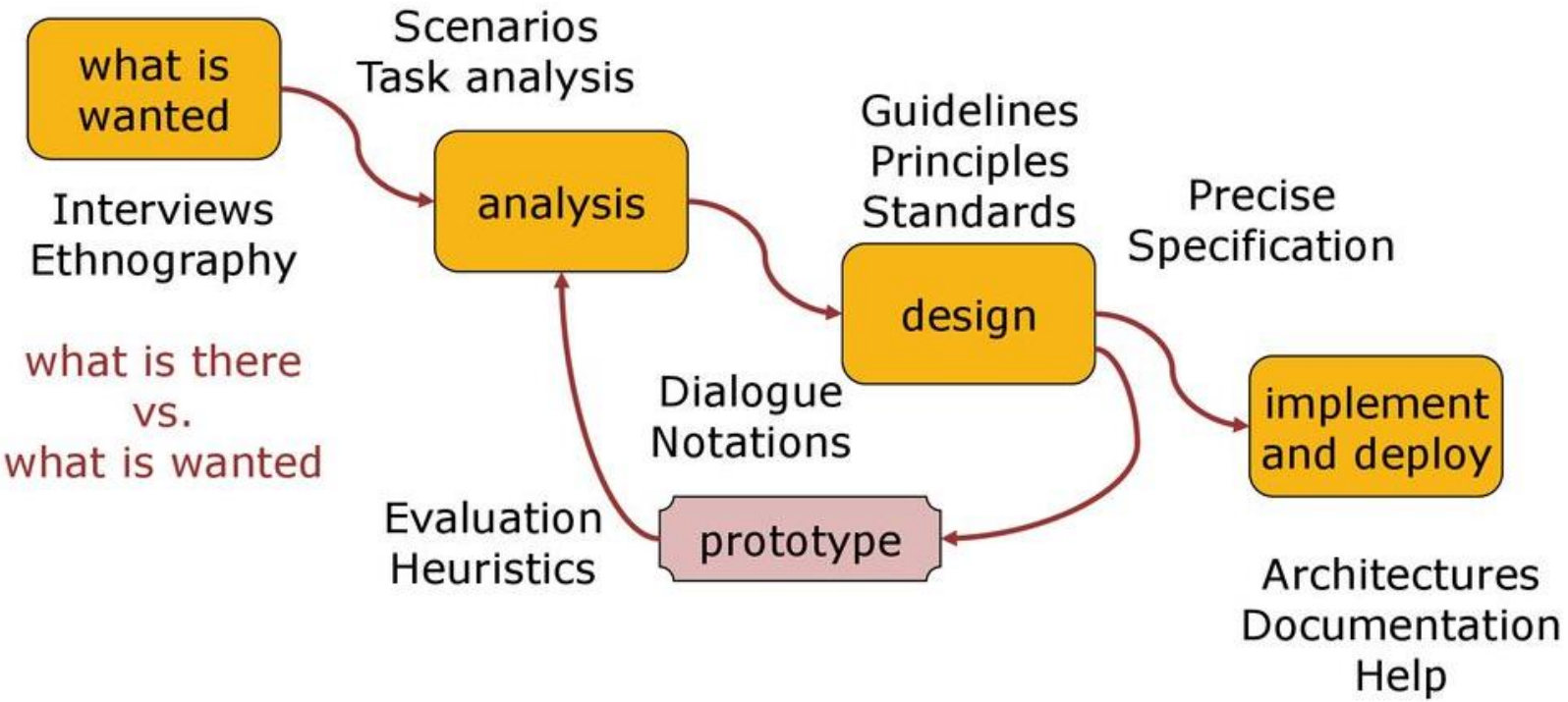
MODULE 7: HUMAN-ROBOT INTERACTION: DESIGN AND BUILD WORKSHOP

Nicholas Ho, PhD

Institute of System Science, NUS

Ideal Process



Application of Process to Workshop Topic: Personal Healthcare Robot

Let's Discuss Together!

1. What is Wanted???
2. Analysis – Scenarios
3. Design (What, Who, Why???)
4. Prototypes
5. Implement and Deploy

Application of Process to Workshop Topic: Personal Healthcare Robot

Group Discussion:

1. What is Wanted??? **– Words/images; Point form**
2. Analysis – Scenarios **– Words/images; Point form**
3. Design (What, Who, Why???) **– Words/images; Point form**
4. Prototypes **– Rough image with labels**
5. Implement and Deploy **– ROS Architecture**

MODULE 7 WORKSHOP: WORKSHOP DAY 4

FINAL PROJECT

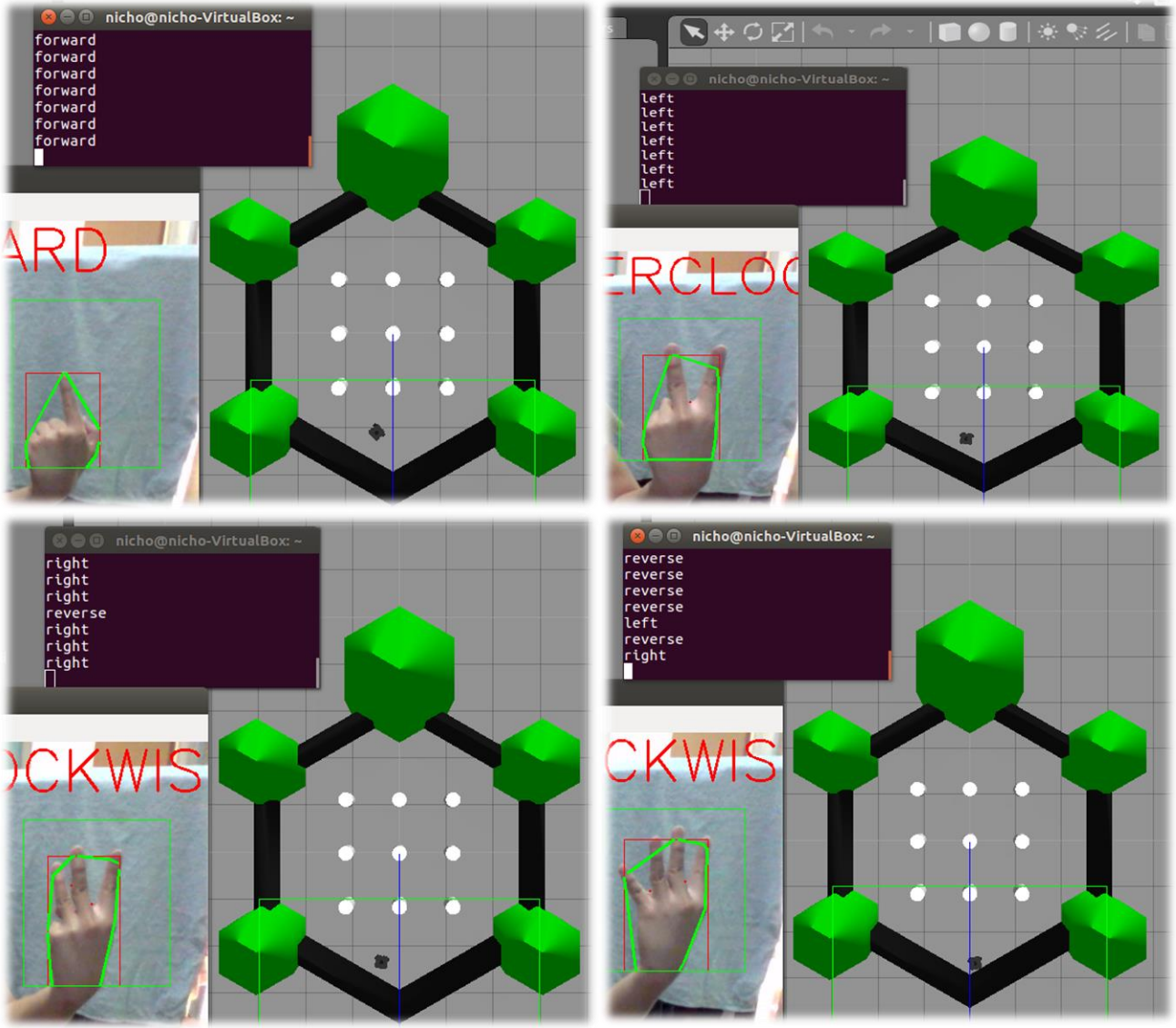
Introducing Hand Gesture Control Robots

Source: https://www.youtube.com/watch?v=KXbXycVTYk0

(A) Applying Hand Gesture Control to TB3 using camera

- **Simple Machine Vision algorithms** (via OpenCV) to recognize various hand gestures; require webcam and preferably blue background
- **Simple finger signs** (1, 2, 3, 4) to control movements of TB3 (Forward, Left, Right, Reverse)
- Using node example: **gesture_control.py**

(A) Applying Hand Gesture Control to TB3 using camera



(A) Applying Hand Gesture Control to TB3 using camera

Setup (install Extension Pack for Vmware); ONLY for VMware users:

- This is required in order to use webcams in VirtualBox guest OS
- Refer to following website on instructions (Summarized below): https://scribles.net/using-webcam-in-virtualbox-guest-os-on-windows-host/
- **All steps to be done in your Windows platform:**
  1. Go to http://download.virtualbox.org/virtualbox/ and download the extension pack which has the same version as your VirtualBox; e.g. my case, my VirtualBox is v6.1.4 so I downloaded this: http://download.virtualbox.org/virtualbox/6.1.4/Oracle_VM_VirtualBox_Extension_Pack-6.1.4.vbox-extpack
  2. Launch “Oracle VirtualBox Manager” and navigate to “File” -> “Preferences”. In ‘Preferences’ window, select ‘Extensions’
  3. Press ‘Add new package’ icon, search the relevant folder and select the downloaded extension pack and install it

(A) Applying Hand Gesture Control to TB3 using camera

Setup for Webcam; ONLY for VMware users:

- Refer to following website on instructions (Summarized below): https://scribles.net/using-webcam-in-virtualbox-guest-os-on-windows-host/
- **All steps to be done in your Windows platform:**
  1. On WINDOWS Command Prompt (search command prompt in search bar in your WINDOWS platform), type the following:

     ```
     cd c:\Program Files\Oracle\VirtualBox
     ```

  2. Next, type the following to list the available cameras:

     ```
     VBoxManage List webcams
     ```

     Take note of the camera that you desire to use (e.g. .1 or .2, etc)

  3. Launch Ubuntu on your VM and wait for it to fully boot
  4. Next, type the following to attach webcam(s) you want to use on your Virtual Machine (**you must ensure that your Ubuntu is launched first on your VM**)

     ```
     VboxManage controlvm "Ubuntu 16.04" webcam attach .x
     ```

     Change "**X**" to the camera no. that you are using!

     Note that you have to do this each time you boot your Ubuntu system on the virtual machine

  5. You may check if the procedures are done properly by searching on Ubuntu for Cheese Webcam Booth application; open it and see if your webcam can be used

(A) Applying Hand Gesture Control to TB3 using camera

Guide to Execute gesture_control.py node:

1. Launch Gazebo in TurtleBot3 World

   ```
   $ roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

2. Run node that allows you to use finger gestures to control the TB3

   ```
   $ rosrun hrse gesture_control.py
   ```

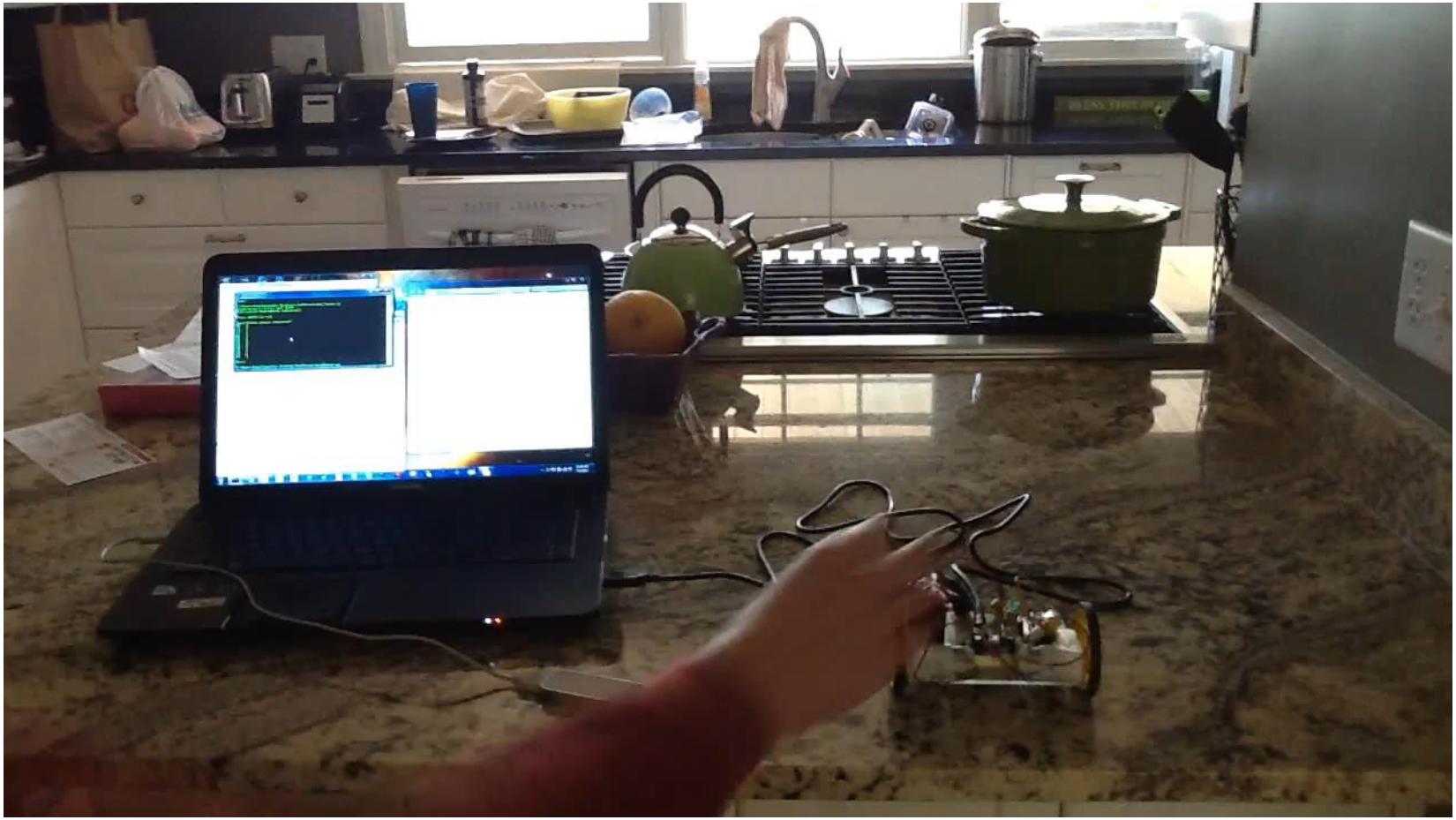
3. Adjust camera until the bounding box is within blue background

4. Try the gesture control by changing your finger gestures into 1, 2, 3 or 4 as illustrated in the demo or pictures; 1, 2, 3, 4 are represented by the following controls:

   ```
   1 = Forward, 2 = Left (Counterclockwise), 3 = Right
   (Clockwise), 4 = Backwards, No Input = Brake
   ```

*****Note that a blue background is strongly recommended for the MV algorithms to work properly**

(B) Applying Hand Gesture Control to TB3 using Leap



Source: https://www.youtube.com/watch?v=r-nTqWW4KaE

(B) Applying Hand Gesture Control to TB3 using Leap

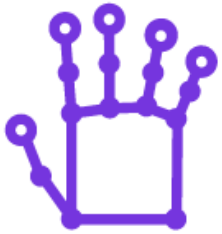
Leap Motion Controller

An optical hand tracking module that captures the movement of users' hands and fingers so they can interact naturally with digital content

Small. Fast. Accurate.
World-class hand tracking

(B) Applying Hand Gesture Control to TB3 using Leap

Key features



Robust and reliable skeletal model

Motion-to-photon latency below the human perception threshold

Development and public use (certified CE, FCC, CAN ICES-3)

(B) Applying Hand Gesture Control to TB3 using Leap

Leap Motion Controller (Hardware)

Specifications

Power supply:	5V DC via USB connector (minimum 0.5A)
Data connection:	USB 2.0 (packaged with USB 2/3 hybrid cable, but can be used with any certified USB standard cables with the Hi-Speed USB 2.0 logo featured on the packaging).
Ingress protection:	Splash resistant
Mounting methods:	May be placed on a desktop, mounted on a VR headset using the Leap Motion VR Developer Mount (https://www.ultraleap.com/product/vr-developer-mount/), or recessed into a larger hardware installation.
Interaction zone:	Depth of up to 60cm (24") preferred, up to 80cm (31") maximum; 140x120° typical field of view. Tracking works in a range of environmental conditions.
Cameras:	Two 640x240-pixel near-infrared cameras; spaced 40 millimetres apart; with infrared-transparent window, operate in the 850 nanometre +/-25 spectral range; typically operates at 120Hz; capable of image capture within 1/2000th of a second.
Camera interface:	Experimental Universal Video Class (UVC) release provides access to low-level controls such as LED brightness, gamma, exposure, gain, resolution, etc.; examples in C, Python, and Matlab, as well as OpenCV bindings.
LEDs:	Three, spaced on either side and between the cameras, baffled to prevent overlaps.
Construction:	Aluminium and scratch-resistant glass
Ambient operating temperature:	32º to 113º F (0º to 45ºC)
Storage temperature:	14º to 122º F (-10º to 50º C)
Relative Humidity:	5% to 85% (non-condensing)
Operating Altitude:	0 to 10,000 feet (0 to 3048 meters)
Compliance:	CE, FCC, CAN ICES-3, REACH, RoHS
Minimum system requirements (desktop):	Windows® 7+ or Mac® OS X 10.7 (note that OSX is no longer formally supported); AMD Phenom™ II or Intel® Core™ i3/i5/i7 processor; 2 GB RAM; USB 2.0 port. VR headsets may come with their own system requirements.

(B) Applying Hand Gesture Control to TB3 using Leap

Leap Motion Controller (Hardware)

(an infrared light based stereoscopic camera)

Cameras:	Two 640x240-pixel near-infrared cameras; spaced 40 millimetres apart; with infrared-transparent window, operate in the 850 nanometre +/-25 spectral range; typically operates at 120Hz; capable of image capture within 1/2000th of a second.
Camera interface:	Experimental Universal Video Class (UVC) release provides access to low-level controls such as LED brightness, gamma, exposure, gain, resolution, etc.; examples in C, Python, and Matlab, as well as OpenCV bindings.
LEDs:	Three, spaced on either side and between the cameras, baffled to prevent overlaps.

The heart of a device is 2 cameras and 3 infrared LEDs

These track infrared light at a wavelength of 850 nanometers, which is outside the visible light spectrum.

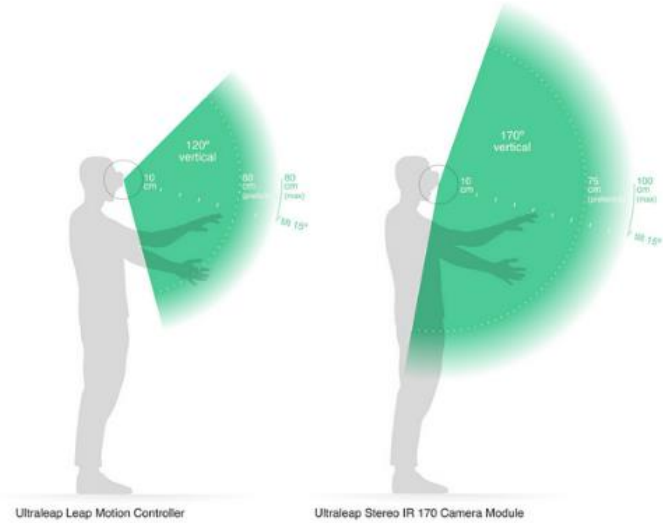
(B) Applying Hand Gesture Control to TB3 using Leap

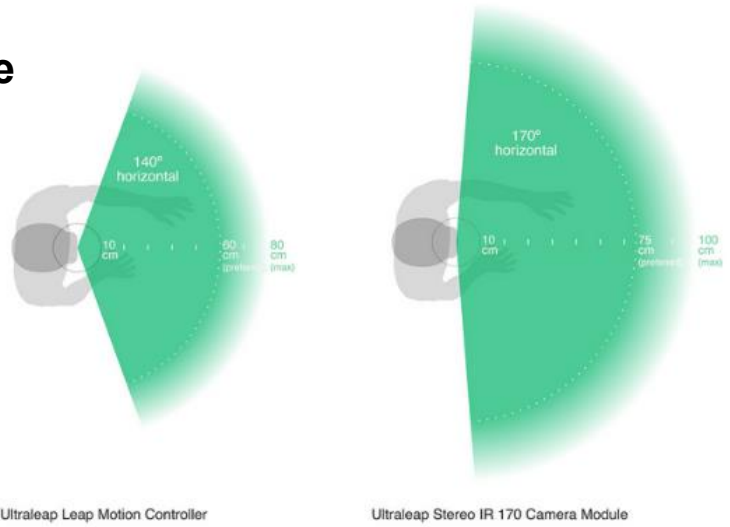
Leap Motion Controller

It **illuminates the close space near to the cameras with infrared light**, hence enabling it to capture a user's hands and fingers

The data takes the form of **a grayscale stereo image of the near-infrared light spectrum, separated into the left and right cameras**. Typically, the **only objects you'll see are those directly illuminated by the device's LEDs**.

However, incandescent light bulbs, halogens, and daylight will also light up the scene in infrared. You might also notice that certain things, like cotton shirts, can appear white even though they are dark in the visible spectrum.





Source: https://www.ultraleap.com/company/news/blog/how-hand-tracking-works/

(B) Applying Hand Gesture Control to TB3 using Leap

Leap Motion Controller (Software)

- The hand tracking platform **doesn't generate a depth map** – instead it **applies advanced algorithms to the raw sensor data**
- After compensating for background objects (such as heads) and ambient environmental lighting, the images are analyzed to reconstruct a 3D representation of what the device "sees"
- Next, the tracking layer matches the data to extract tracking information such as fingers. The hand tracking algorithms interpret the 3D data and infer the positions of obstructed objects. Filtering techniques are applied to ensure smooth temporal coherence of the data

Source: https://www.ultraleap.com/company/news/blog/how-hand-tracking-works/

(B) Applying Hand Gesture Control to TB3 using Leap

Leap Motion Controller

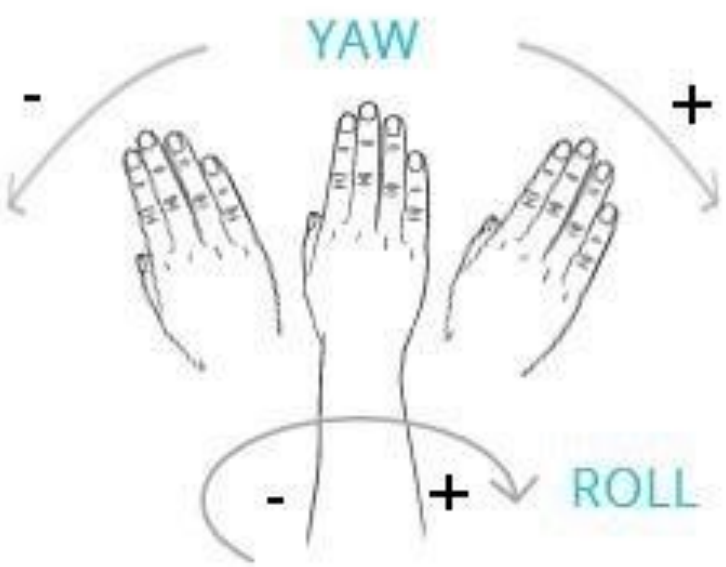
The Leap Motion Controller features a 140×120° field of view and an interactive tracking range of up to 60cm (24") or more.

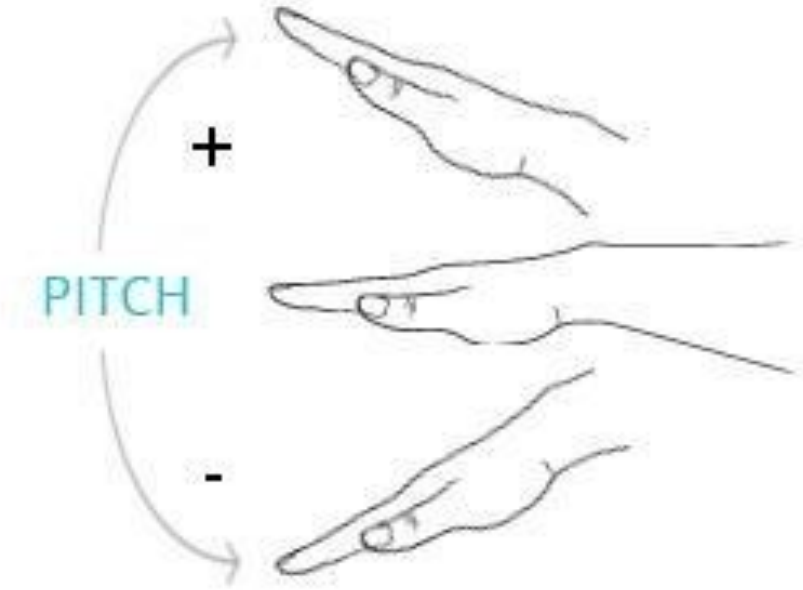
Example applications

- Touchless public interfaces (interactive kiosks, digital out-of-home, elevators)
- Entertainment (location-based VR/AR experiences, arcades, amusement parks)
- Healthcare (stroke rehabilitation, training, mirror, medical imaging, lazy eye treatment)
- Therapy and education (anatomic visualizations, hands-on learning)
- Personnel training (flight simulators, complex computer systems)
- Industrial design and engineering (automotive, assembly lines, facilities management)
- Robotics (telepresence, robotic controls, AI-assisted teaching)
- Remote collaboration

(B) Applying Hand Gesture Control to TB3 using Leap

Leap Motion Controller (Yaw, Pitch, Roll)





(B) Applying Hand Gesture Control to TB3 using Leap

Guide to Execute leap_teleop.py node:

1. Download the leap motion repository from github (if not installed yet) and build the workspace:

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ros-drivers/leap_motion.git
$ cd ~/catkin_ws
$ catkin_make
```

2. Plug in the Leap Motion Controller and activate the control panel:

```
$ LeapControlPanel
```

***Note if LeapControlPanel cannot work, type the following to restart leap:

```
$ sudo service leapd restart
```

(B) Applying Hand Gesture Control to TB3 using Leap

Guide to Execute leap_teleop.py node:

3. Launch Gazebo in TurtleBot3 World

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

4. Run the Publisher node; publish data of your hands' motion to ROS

```
$ rosrun leap_motion sender.py
```

5. Run node that subscribes the data from the previous node and also allows you to use hand gestures to control the TB3

```
$ rosrun hrse leap_teleop.py
```

6. Try the gesture control by changing your hand gestures into the various forms as illustrated in the demo or pictures; these forms are represented by the following:

```
Hands on level (aka starting position):     Brake
Hands (include all fingers) pointing down:  Forward
Hands (include all fingers) pointing up:    Backwards
Hands roll to left:                         Left (Move Counterclockwise)
Hands roll to right:                        Right (Move Clockwise)
```

Group Project 4

In the previous node example (i.e. gesture_control.py), we have utilized simple hand gestures to control the TB3 movements with the help of basic Machine Vision (MV) algorithms. However, this system example is incomplete as it does not account for (a) human errors (e.g. when we went out of control) or (b) inaccurate identifications from the MV; these factors can often lead to unnecessary collisions. One way to rectify this is to adopt automated safety measures such as having obstacle avoidance algorithms to ensure that the TB3 does not collide into the walls or people for example, regardless of the user's gesture inputs.

For this project, we will be editing the node, gesture_control.py to enable an automated safety measure based on a simple obstacle avoidance algorithm. Hence, in the event when we went out of control or due to inaccurate MV identifications, this algorithm will ensure that the TB3 avoids any collisions. **A template (i.e. Project4_TEMPLATE.py) is given to you for this purpose**; you are required to edit 3 Sections of the code to complete it.

Group Project 4

Guide to Execute Project4.py node:

1. Launch Gazebo in TurtleBot3 World

   ```
   $ roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

2. Edit the Project4_TEMPLATE.py (given; 3 Sections to edit) to enable the TB3 to avoid obstacles and rename it as Project4.py

3. Run node that allows you to use finger gestures to control the TB3 while also enabling the TB3 to avoid obstacles

   ```
   $ rosrun hrse Project4.py
   ```

4. Adjust camera until the bounding box is within blue background

5. Change your finger gestures into 1, 2, 3 or 4 as illustrated in the demo or pictures; 1, 2, 3, 4 are represented by the following controls:

   ```
   1 = Forward, 2 = Left (Counterclockwise), 3 = Right
   (Clockwise), 4 = Backwards, No Input = Brake
   ```

6. Check if the node enables the TB3 to avoid obstacles when you went out of control or when the MV has inaccurate identifications

*****Note that a blue background is strongly recommended for the MV algorithms to work properly**

Group Project 4

Hints to complete `clearance()` function:

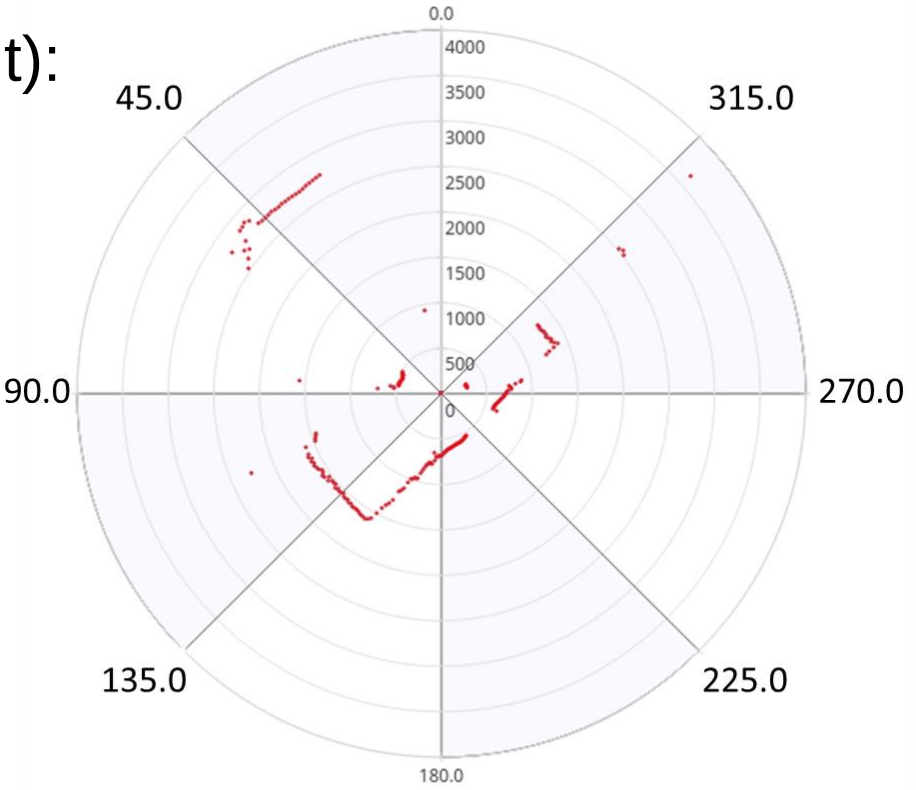
- The purpose of the clearance function is to move the TB3 into a “clear” zone that is not near any obstacles
- Refer to the chart (right):

➢ When obstacles are near front of TB3, it has to move backwards to clear itself

➢ When obstacles are near back of TB3, it has to move front to clear itself

➢ When obstacles are near left of TB3 (90 degrees), it has to move right to clear itself

➢ When obstacles are near right of TB3 (270 degrees), it has to move left to clear itself



Instructions

This is a group project. Each group submits one zip file of your code (i.e. Project4.py) into LumiNUS at the end of the workshop

A123456_A234567_A345678_P4.zip

- Download all files in the directory **/workshops/day4** for reference codes
- Refer to the README file for instructions

OPTIONAL TASK

CUSTOMIZED HAND GESTURE LEAP-BASED CONTROLLER + PHYSICAL TB3 TEST

Group Project 4b (Optional)
(with Leap Motion Controller instead)

Guide to Execute Project4b.py node:

1. Launch Gazebo in TurtleBot3 World

   ```
   $ roslaunch turtlebot3_gazebo turtlebot3_world.launch
   ```

2. Edit `leap_teleop.py` so that it serves the same purpose as Project 4 and rename it as Project4b.py

3. Run node that allows you to use hand gestures to control the TB3 while also enabling the TB3 to avoid obstacles

   ```
   $ rosrun hrse Project4b.py
   ```

3. Change your hand gestures into the various forms as illustrated in the demo or pictures; these forms are represented by the following:

   ```
   Hands on level (aka starting position):     Brake
   Hands (include all fingers) pointing down:  Forward
   Hands (include all fingers) pointing up:    Backwards
   Hands roll to left:                         Left (Move Counterclockwise)
   Hands roll to right:                        Right (Move Clockwise)
   ```

4. Check if the node enables the TB3 to avoid obstacles when you went out of control or when the system has inaccurate identifications

Optional: Your Customized Hand Gesture Controller

Guide to Execute leap_teleop_customized.py node:

1. Launch Gazebo in TurtleBot3 World

   ```
   $ roslaunch turtlebot3_gazebo
   turtlebot3_world.launch
   ```

2. Edit `leap_teleop.py` with a new set of hand gesture controller rules and rename it as leap_teleop_customized.py
3. Run node that allows you to use hand gestures to control the TB3

   ```
   $ rosrun hrse leap_teleop_customized.py
   ```

3. Change your hand gestures into the various forms according to what you have customized
4. Check if the node enables you to control the robot according to what you planned for

Physical TB3 Test
RECAP on Sequences
(Refer to README for detailed instructions)

1. Do the OpenCR Setup (if have not done so)
2. Run roscore and do bring up procedures
3. Run node that allows you to use finger gestures to control the TB3 while also enabling the TB3 to avoid obstacles (i.e. Project4.py)
4. Adjust camera until the bounding box is within blue background
5. Change your finger gestures into 1, 2, 3 or 4 as illustrated in the demo or pictures; 1, 2, 3, 4 are represented by the following controls:

```
1 = Forward, 2 = Left (Counterclockwise), 3 = Right
(Clockwise), 4 = Backwards, No Input = Brake
```

6. Check if the node enables the TB3 to avoid obstacles when you went out of control or when the MV has inaccurate identifications

*****Ensure that your Project4.py work for the virtual system first!!!**

*****Note that a blue background is strongly recommended for the MV algorithms to work properly**

THANK YOU

Email: nicholas.ho@nus.edu.sg