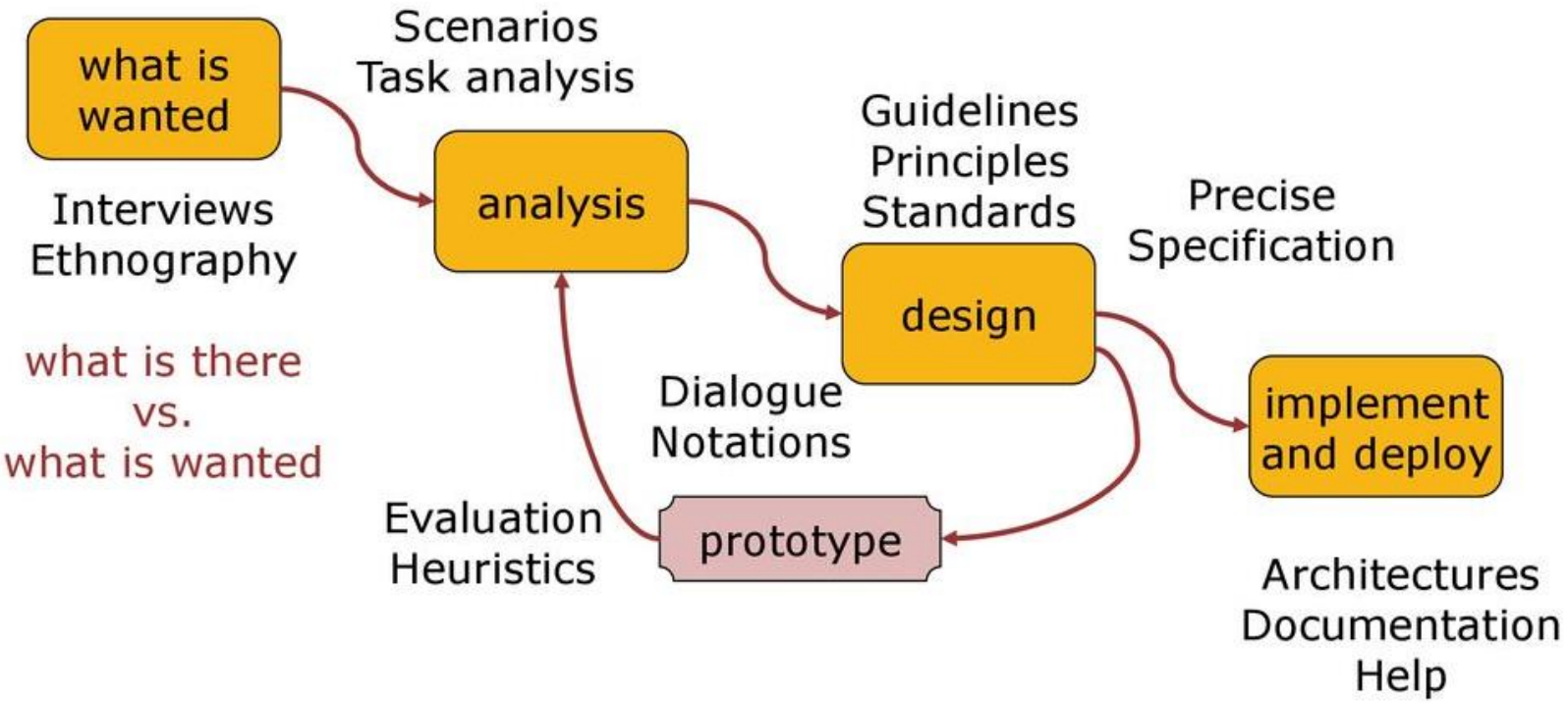
MODULE 7: HUMAN-ROBOT INTERACTION: DESIGN AND BUILD WORKSHOP

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

MODULE 7 WORKSHOP: WORKSHOP DAY 4

FINAL PROJECT

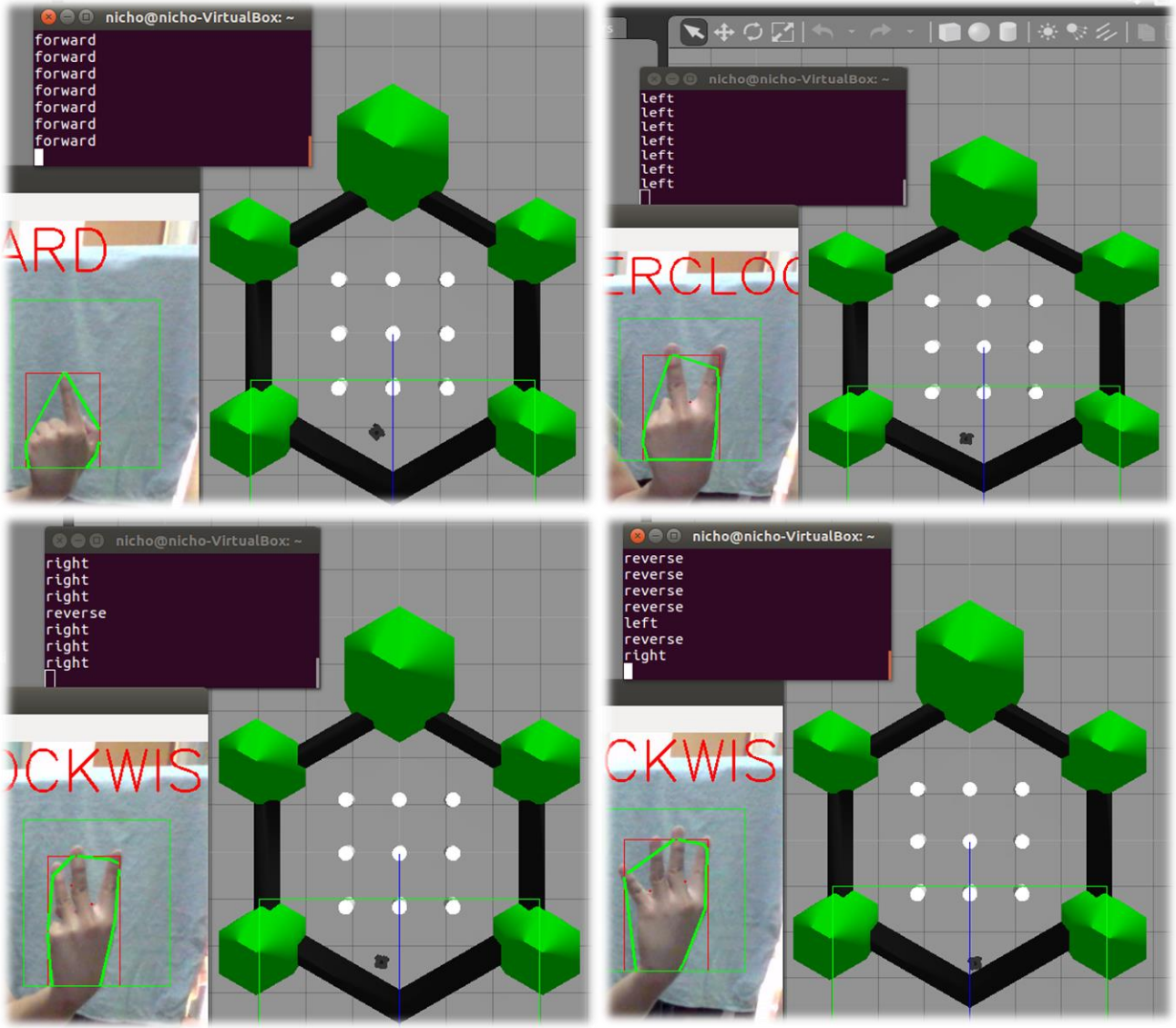
Introducing Hand Gesture Control Robots

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

Applying Hand Gesture Control to TB3

- **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**

Applying Hand Gesture Control to TB3



Applying Hand Gesture Control to TB3

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

Applying Hand Gesture Control to TB3

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

Applying Hand Gesture Control to TB3

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**

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 Project.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:

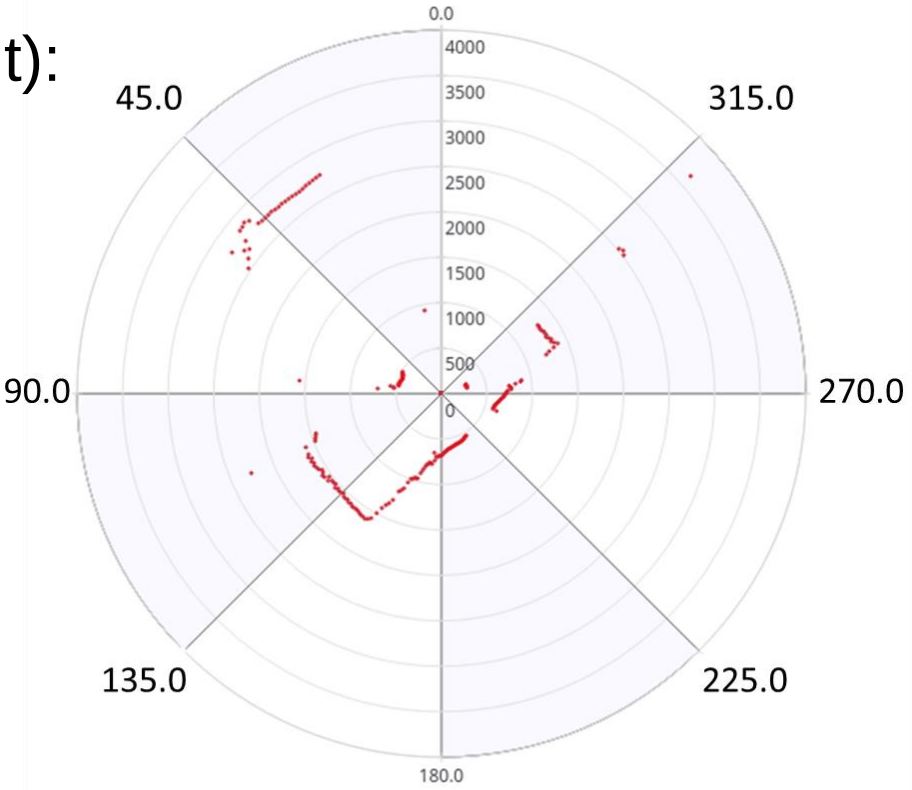
- The purpose of the clearance function is to move the TB3 into a “clear” zone that is not near any obstacles
- Refer to the chart (right):

➢ 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

PHYSICAL TB3 TEST FOR GESTURE CONTROL

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