Technical documentation

version 1.0

Table of Contents

Dockerfile

- Base image: ros:melodic
- Instalations:
  - vim
  - mc
  - ros-melodic-turtlesim
  - ros-melodic-rosbridge-server
  - ros-melodic-tf2-web-republisher
- RUN commands
  - create and build catkin workspace
  - create and build a package ‘turtle_line_cleaner’ with dependencies:
    - geometry_msgs
    - std_msgs
    - std_srvs
    - rospy
  - append lines to CmakeList.txt in ‘turle_line_cleaner’ - to python scrips install and run properly
  - make directory in ‘turtle_line_cleaner’ package ‘scripts’
- COPYcommands
  - *clearService.py* to ‘scripts’ directory in ‘turtle_line_cleaner’ package
  - *container_entrypoint.sh* to /
    - after copy command both files are set as executable
- Entrypoint
  - *container_entrypoint.sh* is used as entrypoint Entrypoint

Launcher

All processes can be run via file *launcher.sh.* This file consists of:

- add a logged user to docker group to use docker commands without sudo
- bild image with name ‘turle-app’
- contaniner use gui application (turtlesim), so display of host computer has to be shared to container: *xhost local:root* gives permission to X sever host necessary to use host display
- *docker run* command is consist of:
  - is run on background – flag *-id*
  - with name ‘turtle-app_container’
  - enviroment variables:
    - “DISPLAY”
    - “QT_X11_NO_MITSHM=1”
  - volume
    - “/tmp/.X11-unix:/tmp/.X11-unix:rw”
  - image ‘turtle-app’
- print to console
- sleep for 20 sec – ensure to all process in container run properly. Then a python application Python application is called
- call *simpleApp.py* - Python application
- after exit *simpleApp.py* the process of container will be killed and removed from container list
- clear console

Entrypoint

Entrypoint is bash file *container_entrypoint.sh.* This file consists of:

- source */ros_entrypoint.sh* which consist of source */opt/ros/melodic/setup.bash* – to allow use ros commands
- source */home/catkin_ws/devel/setup.bash* – source new package (‘turtle_line_cleaner’)
- run *roscore* on background
- run *turlesim turtlesim_node* on background
- run *rosbridge_server* on background
- run *tf2_web_republisher* on background
- run *turtle_line_cleaner clearServise.py*

- run *bin/bash*
- after each command is sleep from 3 to 5 second. It is because each command need some time to start properly.

Service turtle_line_cleaner

File *clearService.py* is copied in Dockerfile to container. It connect to *clear* service with type *std_srvs.srv.Empty*. In script is while loop which runs till condition *rospy.is_shutdown()* is *False*. In while loop is called service *clear* 10 times pre second (*rate=rospy.Rate(10)).* Commands are executed in try and except. When command has an error then an error message is printed to console.

Python application

Name od file is *simpleApp.py* and it is copied in dockerfile to container. Applicatin uses library **roslibpy** to communicate with ros in container and provides user to control turtle via arrows keys. Application is consists of:

- class KeyListener – whole functionality is in this class
    - constructor:
        - get ip address of container with name ‘turtle_app_container’
        - connection to container
        - create a publisher to topic: /*turtle1/cmd_vel* with message type *geometry_msgs/Twist*
        - *q*uick instruction how to use application
        - create listener on events: *on_press, on_release* (keys)
    - *moveXpositive(self)*
        - print information of moving direction
        - publish on topic /*turtle1/cmd_vel* and move turtle in X axis
        - *sefl* - all objects in class can be used in the function
    - *moveXnegative(self), rotateZclockwise(self), rotateZxounterclockwise(self)*
        - similar to *moveXpositive(self)* – in function *rotation...(self)* turtle rotating in Z axis
    - *on_press(sefl, key)* – call when some key is pressed
        - clear console
        - set variable *pressedFlag* to value 1 – only for testing
        - call function corresponding to pressed arrow key in a thread when client is ready
        - *self* – all object in class, *key* – objeckt from library ‘pynput’ where pressed or released key is stored
    - *on_release(self, key)* – call when some key is released

- *when ESC* key is released the application will close
- *self* – all object in class, *key* – objeckt from library ‘pynput’ where pressed or released key is stored