ASA Delivery System

ROS Development Kit

for Keenon W3Pro

1. Asa_charge

- 1.1: Asa_charge_state

A programme created to check the charging status of Keenon W3 Pro.

When called, the programme will subscribe to rostopic “/charge_state_fromSTM32” developed by Keenon Robot.

Return True or False to the server for further action

Code Explain:

```
def charge_status(self):

    global sub

    sub = rospy.Subscriber("/charge_state_fromSTM32", ChargeFB, self.charge_callback)

    while True:

        if self.status != None:

            counter = self.charge_now()


            if not counter:

                sub.unregister()
                return False
            else:
                return True
```

Create a function for subscribing rostopic “/charge_state_fromSTM32” and get the data through the callback

```
def charge_callback(self, data):

    self.data = data

    self.status = data.state
```

from the callback function, the programme still store the state status from charge_state

```
# Define timeout for checking the current state
def charge_now(self):

    timelimit = 40

    start = time.time()

    while True:

        time.sleep(0.1)

        remaining = timelimit + start - time.time()

        print(remaining)

        if self.status == 53:

            sub.unregister()
            return True

        elif remaining <= 0:

            return False
```

the programme will loop until timeout (default setting: 40s) and return to boolean

- **1.2: Asa_charge_task**

A programme created for executing charging instructions while called by the server.

When called, the programme will create an actionlib client and send the requested data to the action server “charge_task” for executing instructions “go_to_charge_pile” and “leave_chrage_pile”.

Return True / False boolean to the server according to the result of the function.

Code Explain:

```
if __name__ == "__main__":

    # create node asa_charge_task
    rospy.init_node("asa_charge_task")

    #create service goCharge for external call
    rospy.Service('/asa_charge_task/goCharge', AsaChargeTask, charge_task)

    rospy.Service('/asa_charge_task/leftCharge', AsaLeftCharge, left_charge)

    rospy.spin()
```

Create a node and create service “goCharge” and “leftCharge” for external call

```
def chargeStatus():

    checkChargeState = charge_state()

    status = checkChargeState.charge_status()

    return status
```

Create a function for calling asa_charge_state to get the data from topic “/charge_state_fromSTM32”

```python
def charge_task(req):

    # create action client to passing the type to the action
    # keenon_charge to the constructor.

    client = actionlib.SimpleActionClient('charge_task', ChargeTaskAction)

    # Waits until the action server has started up and started
    # listening for goals.

    client.wait_for_server()

    # Create a goal to send to the action server.
    goal = ChargeTaskGoal()

    goal.goal_type = 1

    # Create the goal to the action server.

    client.send_goal(goal)

    # Check the charge status of the robot

    status = chargeStatus()

    if not status:
        cancelCharge()
        return False
    else:
        return True
```

Create function “charge_task” as the callback of service “goCharge” with a actionlib client for sending instruction to the robot

Inorder to cancel the charge task, the programme import GoalID from actionlib_msg and publish it to rostopic “charge_task”

```python
def left_charge(req):
    client = actionlib.SimpleActionClient('charge_task', ChargeTaskAction)
    client.wait_for_server()
    goal = ChargeTaskGoal()
    goal.goal_type = 3
    client.send_goal(goal)
    wait = client.wait_for_result()
    if not wait:
        return False
    else:
        return True
```

Create function “left_charge” as the callback of service “leftCharge” which has the same structure of function “charge_task”

2. Asa_delivery

- 2.1: Asa_cancel

A programme created for cancel the action of actionlib server “move_base_action”

This function will be called when the move_base’s mission is failed in following situation:

1. The Goal Location is blocked
2. Run out of mission timeout

Code Explain:

```
class cancelGoal:

    def sendCancel(self):

        self.cancel = rospy.Publisher("/move_base/cancel", GoalID, queue_size=1)

        cancelMsg = GoalID()

        counter = self.cancel.publish(cancelMsg)

        if not counter:

            rospy.logerr("Error: Can't cancel goal.")
            return False

        else:

            rospy.loginfo("Success: Move Base Goal is now canceled.")
            return True
```

Create function “sendCancel” for canceling the goal while the robot cannot arrive the goal position

In order to cancel the move base goal, the program imports GoadID from actionlib_msg, and publishes it to the topic “/move_base/cancel” to clear the move_base_action instruction.

- **2.2: Asa_Enter_lift**

A programme created for instructing the robot to go into the lift's waiting zone.

The programme will check the pose of the robot while the robot successfully arrives at the target zone or fails.

If the robot fails to enter the target waiting zone, the robot will call out the function "asa_exit_lift" automatically and wait for return.

This Function have two return - ifInLift and isInGoal

Code Explain:

```
# Lift_schedule
def enter_lift_schedule(floor):

    isInGoal = False
    isInLift = True

    # create pos_check instant

    tracking = pos_check()

    # Step 1: from waiting zoom enter lift's searching zoom
    inGoal = asa_move_base.asa_move_base(floor, "goal", 25)

    if not inGoal:
        isInGoal = False
    elif inGoal:
        isInGoal = True
        isInLift = tracking.pose_check(floor)


    # Sub Step: if step 1 or step 3 failed, back to lobby waiting zone
    if isInGoal ==  False:
        exitLift , isInLift =  asa_exit_lift.quit_lift(floor)


    # return counter to the server

    return isInGoal, isInLift
```

Create function "enter_lift_schedule" and send the goal's data to "asa_move_base"
While the execution of entering goal position fail or over time limit, call "asa_exit_lift"

- **2.3: Asa_Exit_lift**

A programme created for instructing the robot to exit the lift area.

The programme subscribes to “asa_pose” and will check the pose of the robot while the robot successfully exits the lift or fails.

This Function have two return - ifInLift and isInGoal

Code Explain:

```
def quit_lift(floor):

    exitLift = asa_move_base.asa_move_base(floor, "waiting_zone", 25)

    isInLift = check_pose(floor)

    Goalcancel()

    return exitLift, isInLift
```

Create function “quit_lift” and send the goal’s data to “asa_move_base”

```
# Lift - from inside go to waiting zone
def check_pose(floor):

    tracking = pos_check()

    inLift = tracking.pose_check(floor)

    return inLift

def Goalcancel():

    nullGoal = cancelGoal()

    nullGoal.sendCancel()
```

Subscribe to “Asa_cancel”, cancel move_base_goal while over time limit

- **2.4: Asa_move_base**

A programme created for executing the moving instruction from the server. Providing a point to point moving service.

Code Explain:

```
    # asa_move_base
def asa_move_base(floor, value, timeout):

    # action client
    client = actionlib.SimpleActionClient('move_base', MoveBaseAction)

    client.wait_for_server()

    # loading the data of building
    building_description = load_building_desc()

    # load the goal's pose from the database
    pose = get_pose_data(floor, value, building_description)

    # send goal to move_base
    goal = move(pose)

    client.send_goal(goal)
    wait = client.wait_for_result(timeout=rospy.Duration(timeout))

    if wait == True:

        resultCheck = client.get_result()

        if resultCheck.ret_status == 1:

            return True

        else:

            return False

    else:

        return False
```

Create function “asa_move_base” as the server and create a actionlib client subscribe to ‘move_base’
Send the goal to move_base and wait_for the result

```python
def load_building_desc( ):

    # TODO - replace map folder with DB

    map_folder = os.path.abspath('/asa_delivery_sys/src/asa_delivery_sys/asa_delivery/scripts/testing.yaml')

    with open(map_folder) as f:
        yaml_desc = yaml.safe_load(f)

    return yaml_desc
```

Create function “load_building_desc” to load the data from the database by the target pose data from the server input.

```python
def get_pose_data(floor, goal, database):

    # Create Array

    pose = []

    # load the posistiob data

    x = database[floor][goal]["x"]
    y = database[floor][goal]["y"]
    z = database[floor][goal]["z"]
    w = database[floor][goal]["w"]

    pose.append(
        (
            x,
            y,
            z,
            w
        )
    )

    print(x, y, z, w)

    return pose
```

Create function “get_pose_data” to record the data needed passing to the move_base and store to a list.

```python
def move(pose):

    # Create a new goal with MoveBaseGoal constructor

    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()

    # X,Y,Z coor data

    goal.target_pose.pose.position.x = pose[0][0]
    goal.target_pose.pose.position.y = pose[0][1]
    goal.target_pose.pose.position.z = 0.0

    # X,Y,Z,W orientation

    goal.target_pose.pose.orientation.x = 0.0
    goal.target_pose.pose.orientation.y = 0.0
    goal.target_pose.pose.orientation.z = pose[0][2]
    goal.target_pose.pose.orientation.w = pose[0][3]

    print(goal)

    return(goal)
```

Create a function “move” to make the data from recorded link to the ‘MoveBaseGoal’ type message and return

- **2.5: Asa_pose**

A programme created for checking the current position of the robot. Helps the server determine whether the robot successfully executes the move_base action or not.

If the robot is in the specified position. return True. Else, return False.

Code Explain:

```
# Function - Subscribing to AMCL_POSE
def PoseSub(self):

#     rospy.init_node('pose_sub', anonymous=False)

    # Keep tracking /localization/robot_pose and callback the data
    global sub
    sub = rospy.Subscriber('/localization/robot_pose', PoseWithCovarianceStamped, self.PoseCallBack)
#     rospy.spin()
```

Create function ‘posesub’ that subscribe to the rostopic `'/localization/robot_pose'`

```
# Function - get pose data
def PoseCallBack(self, msg):
    self.msg = msg

    # Position Information from Subscribing  -
    x = msg.pose.pose.position.x
    y = msg.pose.pose.position.y

    self.x_pos = x
    self.y_pos = y

    sub.unregister()
```

The callback of the subscription, helps update the current X,Y position on the map frame.

```python
# check the currrnt pose
def pose_check(self, value):

    self.PoseSub()

    while (True):
        if (self.x_pos != None and self.y_pos != None):
            # TODO - replace map folder with DB

            map_folder = os.path.abspath('/home/peanut/asa_delivery_sys/src/asa_delivery_sys/asa_delivery/scripts/Lift.yaml')

            with open(map_folder) as f:

                yaml_desc = yaml.safe_load(f)

            # get the X coordinate of the zone
            x1 = yaml_desc[value]['x1']
            x2 = yaml_desc[value]['x2']
            x3 = yaml_desc[value]['x3']
            x4 = yaml_desc[value]['x4']

            # get the Y coordinate of the zone
            y1 = yaml_desc[value]['y1']
            y2 = yaml_desc[value]['y2']
            y3 = yaml_desc[value]['y3']
            y4 = yaml_desc[value]['y4']

            print(x1, x2, x3, x4, y1, y2, y3, y4)

            # compare the coordinate with the robot and the zone

            if self.x_pos > x3 and self.x_pos < x2 and self.y_pos > y2 and self.y_pos < y3:

                return True
            else:

                return False
```

Create a function ‘pose_check’ for checking if the robot enters the target zone or not.

- **2.6: Asa_sys**

A programme as the server creates ros service of each function and called by the external end user.

Code Explain:

```
if __name__ == "__main__":

    # create node asa_move_base"
    rospy.init_node("asa_move_base")

    # create service lift_enter for external call
    rospy.Service('/asa_move_base/lift_enter', AsaEnterLift, enter_Lift)

    # create service lift_exit for external call
    rospy.Service('/asa_move_base/lift_exit', AsaExitLift, exit_Lift)

    # create service move_to_pose for external call
    rospy.Service('/asa_move_base/move_to_pose', AsaMoveBase, move_base)

    rospy.loginfo("Lift Control Server started.")

    rospy.spin()
```

Create node “asa_move_base” and service `'/asa_move_base/lift_enter'`,`'/asa_move_base/lift_exit'`,`'/asa_move_base/move_to_pose'`

```
# receive call and send to enter_lift function
def enter_Lift(req):

    print(req)

    enterGoal, enterLift = asa_enter_lift.enter_lift_schedule(req.floor)

    print(enterGoal, enterLift)

    return enterLift, enterGoal
```

With the user’s input, call programme ‘asa_enter_lift’

```
# receive call and send to exit lidt function
def exit_Lift(req):

    exitL, InL = asa_exit_lift.quit_lift(req.floor)

    return InL, exitL
```

With the user’s input, call programme ‘asa_exit_lift’

```
# receive string input and send to asa_move_base server
def move_base(req):

    success = asa_move_base.asa_move_srv(req.floor, req.goal)

    return success
```

With the user's input, call programme 'asa_move_base'

3. Asa_lunach

- **Asa_luanch**

A lunch directory to run the python code inside the package among the workspace “Asa_delivery_sys”

Called by common_robot.launch developed by keenon while the robot booted.

4. Asa_status

- **Asa_status**

This package was created to keep posting the feedback from the subscribed topic.

Code Explain:

```
def __init__(self):

    self.publisher = rospy.Publisher('/asa_status_report', String, queue_size=10)
    self._battery = None
    self._ub_status = None
    self._ch_status = None
```

Create a topic for the server listening.

```
def talker(self):

    rospy.init_node('asa_status')

    rospy.Subscriber('/battery_state', BatteryState, self.battery_callback)

    rospy.Subscriber('/urgency_button_status', Bool, self.ub_callback)

    rospy.Subscriber('/charge_state_fromSTM32', ChargeFB, self.ch_callback)

    rospy.spin()
```

Create a function to subscribe to the topic that will publish the robot status data.

```python
def battery_callback(self, data):

    self._battery = data.percentage

    self.publish_msg()


def ub_callback(self, data):

    self._ub_status = data.data

    self.publish_msg()

def ch_callback(self, data):

    self._ch_status = data.state

    self.publish_msg()
```

Call function “pulish_msg” after ros topic subscribed pulish data to the callback

```python
def publish_msg(self):

    msg = {
        "battery_percentage": self._battery,
        "urgency_button_status": self._ub_status,
        "charge_state": self._ch_status
    }

    msg = json.dumps(msg)

    if (self._battery != None and self._ub_status != None and self._ch_status != None):
        rospy.loginfo(msg)
        self.publisher.publish(msg)
```

publish receive data from the callback to the topic created

5. Asa_switch_floor

- **Asa_switch**

A package created for switching the map of the robot for multiple floor mission.

Code Explain:

```
if __name__ == "__main__":

    # create node asa_switch_map
    rospy.init_node("asa_switch_map")

    #create service goCharge for external call
    rospy.Service('/asa_switch_map/switchFloor', AsaSwitchFloor, switch_map)

    rospy.spin()
```

Created function “switch_map” which subscribes to service `'/switch_dest_floor_map'` developed by keenon.

```
def get_pose_tag(floor):

    map_folder = os.path.abspath('/asa_delivery_sys/src/asa_delivery_sys/asa_delivery/scripts/testing.yaml')
    with open(map_folder) as f:
        yaml_desc = yaml.safe_load(f)

    pose_tag = Pose()

    pose_tag.position.x = yaml_desc[floor]["goal"]["x"]
    pose_tag.position.y = yaml_desc[floor]["goal"]["y"]
    pose_tag.orientation.z = yaml_desc[floor]["goal"]["z"]
    pose_tag.orientation.w = yaml_desc[floor]["goal"]["w"]

    return pose_tag
```

Created function “get_pose_tag” that stored the location data of target floor from database.

```
def switch_map(req):

    # create service proxy to call the keenon switch map

    switch_map_service = rospy.ServiceProxy('/switch_dest_floor_map', SwitchMap)

    # get the pose tag from the yaml file

    pose_tag_current = get_pose_tag(req.currentFloor)
    pose_tag_next = get_pose_tag(req.nextFloor)

    # call the keenon switch map service with the floor and pose tags

    result = switch_map_service(req.nextFloor, pose_tag_current, pose_tag_next)

    return result.result
```

Execute switching floor and localize the robot on the new floor map.

6. How to implement the package

Step 1: Create a workspace directory

- Open Terminal
- From the location you want, type " mkdir 'your_workspace_name' "
- enter the directory you create by typing " cd 'your_workspace_name' "
- create a file call src by typing " mkdir src "
- typing "catkin_make" to build the wordspace

Step 2: Download the package

- Download the repository's zip file from github
  url: https://github.com/ASARobotics/keenon_robot_ROS_system

- Unzip file and enter directory "backup_code"
- Unzip file "Keenon.zip"
- from the directory, copy directory "asa_delivery_sys" to the src directory you have created in the workspace
- back to the top of your workspace and type in "catkin_make"

Step 3: Move the required file to the Ros Directory

- Move all the file in the devel directory to the Keenon Ros Directory
  - the location should be: /opt/ros/indigo/
- From this content with matching the name of different directory
- For the "share" directory, copy directory "asa_delivery_sys" from the src file first, then copy the cmake directory from each package within the "devel/share" from your workspace, and delete the "cmake.txt" file

Step 4: Change the Launch File

- Enter directory "/opt/ros/indigo/share/robot_settings/launch"
- backup the launch file "robot_common.launch" by typing "cp robot_common.launch robot_common.launch.bak"
- Edit the launch file by typing "nano robot_common.launch"
- implement the following code": <include file="/opt/ros/indigo/share/asa_delivery_sys/asa_launch/launch/asa_sys.launch">
  </include> "
- Reboot the robot