amcl_demo.launch

```
<launch>
  <!-- 3D sensor -->
  <arg name="3d_sensor" default="$(env TURTLEBOT_3D_SENSOR)"/>  <!-- r200,
kinect, asus_xtion_pro -->
  <include file="$(find turtlebot_bringup)/launch/3dsensor.launch">
    <arg name="rgb_processing" value="false" />
    <arg name="depth_registration" value="false" />
    <arg name="depth_processing" value="false" />

    <!-- We must specify an absolute topic name because if not it will be
prefixed by "$(arg camera)".
         Probably is a bug in the nodelet manager:
https://github.com/ros/nodelet_core/issues/7 -->
    <arg name="scan_topic" value="/scan" />
  </include>

  <!-- Multi Map server -->
  <arg name="map_file" default="$(env TURTLEBOT_MAP_FILE)"/>
  <node name="multi_map_server" pkg="multi_map_server" type="multi_map_server"
args="$(arg map_file)" />

  <!-- AMCL -->
  <arg name="custom_amcl_launch_file" default="$(find
turtlebot_navigation)/launch/includes/amcl/$(arg 3d_sensor)_amcl.launch.xml"/>
  <arg name="initial_pose_x" default="0.0"/> <!-- Use 17.0 for willow's map in
simulation -->
  <arg name="initial_pose_y" default="0.0"/> <!-- Use 17.0 for willow's map in
simulation -->
  <arg name="initial_pose_a" default="0.0"/>
  <include file="$(arg custom_amcl_launch_file)">
    <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
    <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
    <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
  </include>

  <!-- Move base -->
  <arg name="custom_param_file" default="$(find turtlebot_navigation)/param/$
(arg 3d_sensor)_costmap_params.yaml"/>
  <include file="$(find
turtlebot_navigation)/launch/includes/move_base.launch.xml">
    <arg name="custom_param_file" value="$(arg custom_param_file)"/>
  </include>

</launch>
```

clearMap.py

```
#!/usr/bin/env python

import rospy
import std_srvs.srv
import dynamic_reconfigure.client #this lets you reconfigure node parameters

#This code is written with some guidance from Dr. Eric Schneider concerning
obstacle layer reconfigurement


#cleans the obstacle layer of the map in two different ways
class mapCleaner:
    def __init__(self):
```

```
        self.srv_name = "/move_base/clear_costmaps"  # this is the full path
name of the service
        self.clear_costmaps = rospy.ServiceProxy(self.srv_name,
std_srvs.srv.Empty)

        self.static_layer = "/move_base/global_costmap/static_layer"
        self.static_layer_enabled = "{0}/enabled".format(self.static_layer)
        self.obs_layer = "/move_base/global_costmap/obstacle_layer"
        self.obs_layer_enabled = "{0}/enabled".format(self.obs_layer)

        self.static_client =
dynamic_reconfigure.client.Client(self.static_layer, timeout=5)
        self.obs_client = dynamic_reconfigure.client.Client(self.obs_layer,
timeout=5)

        self.r = rospy.Rate(5)

#clears obstacles in robot's laser radius
    def clearMapClient(self):
        try:
            rospy.wait_for_service(self.srv_name)
            self.clear_costmaps()
            return True
        except rospy.ServiceException:
            rospy.logerr("{0} service call failed: {1}".format(self.srv_name))
            return False

#reconfigures the obstacle layer by switching it on or off
    def set_layers(self, enabled):
        self.static_client.update_configuration({"enabled": enabled})
        # Wait until the parameter has been set
        rospy.Duration(2)

        self.obs_client.update_configuration({"enabled": enabled})
        # Wait until the parameter has been set
        rospy.Duration(2)

        return True

#resets the map to its initial state by turning off and on the obstacle layer
    def reset_costmap_layers(self):
        self.set_layers(False)
        self.set_layers(True)
        return True
'''
if __name__ == "__main__":
    clearMap = mapCleaner()
    #clearMap.clearMapClient() # this is a small clear for the area around a
robot
    #full cleanser:
    success = clearMap.reset_costmap_layers() # full map cleanser
'''
```

__init__.py

```
__all__ = ['clearMap', 'landmarks', 'repose_node', 'simple_navigation_node',
'teleport_node']
```

landmarks.py

```
#this file just contains dictionaries of location to coordinates for
navigational use
```

```
dict5N = {
    "505N" : [18.4, 8.8],
    "517N" : [27, 16],
    "519N" : [30, 18.7],
    #right and left
    "518N" : [26.662, 18.861],
    "517N" : [26.662, 18.861],
    "516N" : [24.39, 18.555],
    #right and left
    "514N" : [21.2, 18.7],
    "515N" : [21.2, 18.7],
    #right and left
    "513N" : [17.845, 18.753],
    "510N" : [17.845, 18.753],
    #front and right
    "511N" : [14.561, 18.632],
    "512N" : [14.561, 18.632],
    "509N" : [15.343, 13],
    #right and front
    "508N" : [15.12, 8.734],
    "507N" : [15.12, 8.734],
    "506N" : [17, 8.53],
    "505N" : [19.33, 8.5],
    #right and left
    "503N" : [21, 8.5],
    "504N" : [21, 8.5],
    "502N" : [23.68, 8.624],
    "501N" : [23.325, 8.5],
    "AWAITEXIT" : [31.3, 13.457],
    "RETREAT" : [30, 13.6]

}

dict5L = {
    "DTOILET" : [39, 21.6], #WATER FOUNTAIN
    "AWAITN" : [33.23, 13.15],
    "MTOILET" : [33.8, 11.5],
    "WTOILET" : [42, 11.5],
    "AWAITS" : [42.166, 13.15],
    "ELEVATOR" : [35, 22.4], # this one probably needs some help
    "AWAITTELE" : [36.5,22.4],
    "RETREAT" : [37.5, 12.2]
}

dict5S = {
    "502S" : [49.15, 17.915],
    "503S" : [50.9, 18],
    "504S" : [51.69, 18],
    "506S" : [53.368, 17.6],
    "505S" : [54.8, 17.8],
    "507S" : [58.2, 18],
    "508S" : [61, 18.2],
    "509S" : [62.94, 20.12],
    "521S" : [64, 20.168],
    "510S" : [67.7, 20.55],
    #front and left
    "512S" : [69, 20.23],
    "511S" : [69, 20.23],
    "513S" : [68.8, 18.642],
    "514S" : [69.28, 14.28],
    "515S" : [69, 11.466],
    "516S" : [68.8, 8.265],
```

```
    #in crawl space
    "517S" : [68.9, 7.75],
    "518S" : [68.9, 7.75],
    "519S" : [68.9, 7.75],
    "520S" : [65.133, 8.142],
    "522S" : [59.565, 8.385],
    "523S" : [56, 7.915],
    "AWAITEXIT" : [44.3, 13.445],
    "RETREAT" : [45, 14]
}

#-------------------------------------------6

dict6L = {
    "ELEVATOR" : [34.979,21.88],
    "AWAITTELE" : [36.455, 21.88],
    "DTOILET" : [39.3, 22],
    "AWAITN" : [31.6, 12],
    "AWAITS" : [40.7, 12],
    "MTOILET" : [32.2, 11.67],
    "WTOILET" : [41.965, 11.67],
    "RETREAT" : [36, 10.5]
}

dict6N = {
    "611N" : [26.3, 13.2],
    "601N" : [23,8, 6.8],
    #left and right
    "602N" : [20.5, 6.8],
    "603N" : [20.5, 6.8],
    #left right
    "604N" : [18.3, 6.8],
    "605N" : [18.3, 6.8],
    #front lef
    "607N" : [14.8, 6.8],
    "606N" : [14.8, 6.8],
    "608N" : [14,8, 13],
    "610N" : [14.8, 17.4],
    "609N" : [13, 17.3],
    "AWAITEXIT" : [29, 12],
    "RETREAT" : [28, 12.4],
}

dict6S = {
    #LEFT RIGHT
    "601S" : [44, 12],
    "603S" : [44, 12],
    "602S" : [58, 12],
    "AWAITEXIT" : [43, 12],
    "RETREAT" : [45, 12.2],
}

#-------------------------------------------7

dict7L = {
    "ELEVATOR" : [33.4, 20.5],
    "AWAITTELE" : [35.3, 20.8],
    "DTOILET" : [37.5, 20.8],
    "MTOILET" : [32, 10],
    "WTOILET" : [41, 10],
    "AWAITS" : [41, 12],
    "AWAITN" : [31.7, 12],
    "RETREAT" : [36, 11],
}
```

```
dict7N = {
    "701N" : [25.8, 16],
    #FRONT RIGHT
    "719N" : [25.8, 17],
    "720N" : [25.8, 17],
    #RIGHT LEFT
    "717N" : [23.4, 17],
    "718N" : [23.4, 17],
    #RIGHT LEFT
    "715N" : [20.1, 17],
    "716N" : [20.1, 17],
    "714N" : [18, 17],
    "709N" : [11.8, 16],
    "713N" : [11.6, 14],
    "712N" : [11.6, 11],
    "711N" : [11.6, 8.5],
    "708N" : [11.6, 6.5],
    "707N" : [13, 6.5],
    #RIGHT LEFT
    "706N" : [17, 6.5],
    "705N" : [17, 6.5],
    #RIGHT LEFT
    "704N" : [19, 6.5],
    "703N" : [19, 6.5],
    #RIGHT LEFT
    "702N" : [23.5, 6.5],
    "701N" : [23.5, 6.5],
    "AWAITEXIT" : [29, 12],
    "RETREAT" : [28, 12.2],
}

dict7S = {
    "AWAITEXIT" : [43, 12],
    #LEFT RIGHT
    "701S" : [44.2, 12],
    "706S" : [44.2, 12],
    "702S" : [54, 12],
    "703S" : [61.7, 12],
    "704S" : [62.5, 7.3],
    "705S" : [62.5, 7.3],
    "RETREAT" : [45, 12.2],
}

dictdict = {
    "dict5N": dict5N,
    "dict5L": dict5L,
    "dict5S": dict5S,
    "dict6N": dict6N,
    "dict6L": dict6L,
    "dict6S": dict6S,
    "dict7N": dict7N,
    "dict7L": dict7L,
    "dict7S": dict7S,
}
```

repose_node.py

```
# !/usr/bin/env python

import rospy
from geometry_msgs.msg import PoseWithCovarianceStamped
```

```
from tf.transformations import quaternion_from_euler

#this code a modified version of The Construct's tutorial, available at
#http://www.theconstructsim.com/ros-qa-140-how-to-modify-a-robots-coordinates-
when-it-arrives-at-a-checkpoint/

#modifies a robot's perceived coordinates by publishing new coordinates
class poseSetter:
    def __init__(self):
        #rospy.init_node('the_only_node', anonymous=True)
        self.pub = rospy.Publisher('/initialpose', PoseWithCovarianceStamped,
queue_size=10)
        rospy.sleep(2)
        self.newPos = PoseWithCovarianceStamped()

        self.newPos.pose.pose.position.x = 0.0
        self.newPos.pose.pose.position.y = 0.0
        self.newPos.pose.pose.position.z = 0.0

        [x, y, z, w] = quaternion_from_euler(0.0, 0.0, 0.0)
        self.newPos.pose.pose.orientation.x = x
        self.newPos.pose.pose.orientation.y = y
        self.newPos.pose.pose.orientation.z = z
        self.newPos.pose.pose.orientation.w = w

#modifies a robot's perceived coordinates by publishing new coordinates
    def newPose(self, nx, ny):
        self.newPos.pose.pose.position.x = nx
        self.newPos.pose.pose.position.y = ny

        self.pub.publish(self.newPos)
'''
if __name__ == "__main__":
    poser = poseSetter()
    poser.newPose(33.4, 20.5)
'''
```

simple_navigation_node.py

```
#!/usr/bin/env python

'''
Copyright (c) 2015, Mark Silliman
All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation and/or
other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
```

```
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''

import rospy
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, Point, Quaternion, Twist
from math import radians
#this is an expanded version of Mark Silliman's Learn Turtlebot Tutorial
available at
#https://learn.turtlebot.com/2015/02/03/11/

#publishes a goal to the navigation stack, returns if the goal was met or not
class GoToPose:
    def __init__(self):
        self.goal_sent = False

        # What to do if shut down (e.g. Ctrl-C or failure)
        rospy.on_shutdown(self.shutdown)

        # setting publisher
        rospy.loginfo("setting publisher")
        self.pub = rospy.Publisher('cmd_vel_mux/input/navi', Twist,
queue_size=10)
        # 5 hz
        r = rospy.Rate(5)

        spin_local = Twist()
        spin_local.linear.x = 0
        spin_local.angular.z = radians(90)

        # Tell the action client that we want to spin a thread by default
        self.move_base = actionlib.SimpleActionClient("move_base",
MoveBaseAction)
        rospy.loginfo("Wait for the action server to come up")

        # Allow up to 5 seconds for the action server to come up
        self.move_base.wait_for_server(rospy.Duration(5))

    def goto(self, pos, quat, actionTime):
        # Send a goal
        self.goal_sent = True
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()
        goal.target_pose.pose = Pose(Point(pos['x'], pos['y'], 0.000),
                                     Quaternion(quat['r1'], quat['r2'],
quat['r3'], quat['r4']))
        # Start moving
        self.move_base.send_goal(goal)

        # Allow TurtleBot up to 60 seconds to complete task
        success = self.move_base.wait_for_result(rospy.Duration(actionTime))

        state = self.move_base.get_state()
        result = False

        if success and state == GoalStatus.SUCCEEDED:
            # We made it!
```

```
            result = True
        else:
            self.move_base.cancel_goal()

        self.goal_sent = False
        return result

#^C behaviour
    def shutdown(self):
        self.pub.publish(Twist())
        if self.goal_sent:
            self.move_base.cancel_goal()
        rospy.loginfo("Stop")

#1440 degree spin
    def makelargespin(self):
        r = rospy.Rate(5)

        spin_local = Twist()
        spin_local.linear.x = 0
        spin_local.angular.z = radians(91)
        for x in range(0, 90):
            self.pub.publish(spin_local)
            r.sleep()

#approximate 720 spin
    def makesmallspin(self):
        r = rospy.Rate(5)

        spin_local = Twist()
        spin_local.linear.x = 0
        spin_local.angular.z = radians(91)
        for x in range(0, 45):
            self.pub.publish(spin_local)
            r.sleep()

#aproximate 180 spin
    def halfMoon(self):
        r = rospy.Rate(6)
        spin_local = Twist()
        spin_local.linear.x = 0
        spin_local.angular.z = radians(45)
        for x in range(0, 25):
            self.pub.publish(spin_local)
            r.sleep()


# waits 4 seconds then moves forward 1.5m
    def awaitDoor(self):
        self.goal_sent = True
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()
        goal.target_pose.pose.position.x = 1.5  # 1.5 meters
        goal.target_pose.pose.orientation.w = 1.0  # go forward
'''
if __name__ == '__main__':
    try:
        # will spin a few times to localize
        navigator = GoToPose()
        #navigator.makesmallspin()

        # Customize the following values so they are appropriate for your
location
```

```
        xText = raw_input("x Coordinate: ")
        yText = raw_input("y Coordinate: ")
        actionTime = raw_input("time to reach goal: ")
        position = {'x': float(xText), 'y': float(yText)}
        quaternion = {'r1': 0.000, 'r2': 0.000, 'r3': 1.000, 'r4': 0.000}

        #position = {'x': 30.7, 'y': 12.5}
       # quaternion = {'r1': 0.000, 'r2': 0.000, 'r3': 1.000, 'r4': 0.000}

        rospy.sleep(1)
        rospy.loginfo("Go to (%s, %s) pose", position['x'], position['y'])
        success = navigator.goto(position, quaternion, 60)

        if success:
            rospy.loginfo("GOAL REACHED")
        else:
            rospy.loginfo("GOAL FAILED")

        # Sleep to give the last log messages time to be sent
        rospy.sleep(1)

    except rospy.ROSInterruptException:
        rospy.loginfo("Ctrl-C caught. Quitting")
'''
```

teleport_node.py

```
#!/usr/bin/env python

import rospy
import multi_map_server.srv

#allows controller call to the set_map_filename service in the multi_map_server
class Teleporter:
    def __init__(self):
        self.fifth = "/home/shu/maps/5thedited.yaml"
        self.sixth = "/home/shu/maps/6thedited.yaml"
        self.seventh = "/home/shu/maps/7thedited.yaml"

    def set_map_filename_client(self, floor):
        fileName = ""
        if floor == 5:
            fileName = self.fifth
        if floor == 6:
            fileName = self.sixth
        if floor == 7:
            fileName = self.seventh
        rospy.wait_for_service('set_map_filename')
        try:
            set_map_filename = rospy.ServiceProxy('set_map_filename',
multi_map_server.srv.set_map_filename)
            response = set_map_filename(fileName)
            return response.success
        except rospy.ServiceException:
            rospy.logerr("{0} service call failed: {1}".format(self.srv_name))

'''
if __name__ == "__main__":
      teleporter = Teleporter()
      success = teleporter.set_map_filename_client(6)
      rospy.loginfo("do tell if it works")
```

'''

controller_framework_node.py

```
#!/usr/bin/env python

import importlib
# NECESSARY IMPORTS
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import actionlib
import rospy
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, Point, Quaternion
# LOCATION DICTIONARY
#EXTRAS
import os
import smtplib
from email.mime.text import MIMEText

from controller_framework import *

class Controller:
    def __init__(self):
        #node
        rospy.init_node('the_only_node', anonymous=True)

        self.nav = simple_navigation_node.GoToPose()
        self.repose = repose_node.poseSetter()
        self.cleaner = clearMap.mapCleaner()
        self.teleporter = teleport_node.Teleporter()
        self.landmarks = landmarks

        #information UPDATE ACCORDINGLY
        self.floor = 7
        self.floorPortion = "L"

        self.inUse = False
        self.elevatorTime = False
        self.inEvel = False
        self.currentGoalPath = []
        self.currentGoal = ""

    #should call to handle moving in and out of the elevator
    def evelEvent(self):
        os.system("espeak 'please press the elevator button for me'")
        os.system("espeak 'please tell me when to enter the elevator by pressing
E'")
        xText = raw_input("press E to Enter/Exit an elevator: ")
        index = self.currentGoalPath.index("ELEVATORTIME")

        rightQuat = [0, 0, 0, 1]
        leftQuat = [0, 0, 1, 0]

        interest = self.currentGoalPath[index + 1]
        success = self.attempt3(interest, rightQuat, 30)
        if not success:
            rospy.loginfo("cant seem to get in the elevator, asking for
help....")
            os.system("espeak 'can't seem to get in the elevator'")
            os.system("espeak 'seeking help'")
            self.askForHelp(False)
            return success
```

```
        os.system("espeak 'please press the floor button'")
        self.teleporter.set_map_filename_client(float(self.currentGoal[0]))
#HOPEFULLY THIS WORKS
        goalDict = "dict" + self.currentGoal[0] + "L"
        pos = self.landmarks.dictdict[goalDict]["ELEVATOR"]
        self.repose.newPose(pos[0], pos[1])

        os.system("espeak 'please tell me when to exit the elevator by pressing
E'")
        xText = raw_input("press E to Enter/Exit an elevator: ")

        self.cleaner.reset_costmap_layers()
        index = index + 2
        interest = self.currentGoalPath[index]
        success = self.attempt3(interest, leftQuat, 30)
        if not success:
            rospy.loginfo("cant seem to get in the elevator, asking for
help....")
            os.system("espeak 'can't seem to get out of the elevator'")
            os.system("espeak 'seeking help'")
            self.askForHelp(True)
            return success

            index = index + 1
            self.go(index, len(self.currentGoalPath))

        self.go(index + 1, len(self.currentGoalPath))
        return success

    #should get a generated route and perform the whole route if its on the same
floor
    #else the elevator logic should be completed by evelEvent
    def goFirst(self, room):
        os.system("espeak 'going to goal'")
        self.inUse = True
        route = self.generateRoute(room)
        print(route)

        if False in route:
            rospy.loginfo("Invalid Room Entered!")
            os.system("espeak 'invalid room entered'")
            os.system("espeak 'please enter room in format floor, room number,
wing'")
            self.inUse = False
            return False

        self.currentGoalPath = route
        self.currentGoal = room

        #chop at the elevator
        if "ELEVATORTIME" in self.currentGoalPath:
            completion = route.index("ELEVATORTIME")
        else:
            completion = len(route)
        self.go(0, completion)

    def go(self, start, completion):
        #complete route generated
        rightQuat = [0, 0, 0, 1]
        leftQuat = [0, 0, 1, 0]
        success = True
        index = 0
        while index < completion and success:
            interest = self.currentGoalPath[index]
```

```
            index = index + 1

            if not isinstance(interest, str) and len(interest) != 1:
                quat = leftQuat
                if index < (len(self.currentGoalPath) - 1):
                    if (self.currentGoalPath[index + 1] == "SLEEP") and
(self.floorPortion == "N" or "L"):
                        quat = rightQuat
                success = self.attempt3(interest, quat, 60)
                continue
            elif len(interest) == 1:
                self.floorPortion = interest
                print("portion is " + self.floorPortion)
            elif interest == "SLEEP":
                os.system("espeak 'please open the door for me'")
                rospy.sleep(5)
                self.cleaner.clearMapClient() #potentially needs a full cleaner

        if not success:
            failure = self.currentGoalPath[index-1]
            rospy.loginfo("had trouble going to " + str(failure) + " seeking
help...")
            os.system("espeak 'had trouble reaching the goal'")
            os.system("espeak 'seeking help'")
            os.system("espeak 'returning to help point'")
            self.askForHelp(False)
            currentDict = "dict" + str(self.floor) + str(self.floorPortion)
            loc = self.landmarks.dictdict[currentDict]["RETREAT"]
            failure = self.attempt3(loc, leftQuat, 60)

        #clean up at end
        if completion == len(self.currentGoalPath) and success: #so no elevator
stuff happened
            self.currentGoalPath == []
            self.currentGoal = ""
            os.system("espeak 'We have arrived at our destination'")
            rospy.sleep(2)
            os.system("espeak 'returning to help point'")
            currentDict = "dict" + str(self.floor) + str(self.floorPortion)
            loc = self.landmarks.dictdict[currentDict]["RETREAT"]
            failure = self.attempt3(loc, leftQuat, 60)
            rospy.loginfo("localizing and clearing costmaps")
            os.system("espeak 'localising and clearing costmaps'")
            self.nav.makesmallspin()
            self.cleaner.reset_costmap_layers()
        return success

    #given a goal generate a path
    def generateRoute(self, goal):
        goalRoute = []  # give it location
        robLobbyDict = "dict" + str(self.floor) + "L"
        robcurrentDict = "dict" + str(self.floor) + self.floorPortion
        goalDict = "dict" + goal[0] + goal[3]  # which dictionary the goal
should be located

        #lobby requests
        if len(goal) != 4 and goal not in self.landmarks.dict5L:
            goalRoute.append(False)
            return goalRoute
        if len(goal) == 4 and goal not in self.landmarks.dictdict[goalDict]:
            goalRoute.append(False)
            return goalRoute
        if len(goal) != 4:
            if self.floorPortion == "L":
```

```
                goalRoute.append(self.landmarks.dictdict[robLobbyDict][goal])
                return goalRoute
            else:
                goalRoute.append(self.landmarks.dictdict[robcurrentDict]
["AWAITEXIT"])
                goalRoute.append("SLEEP")
                goalRoute.append(self.landmarks.dictdict[robLobbyDict]["AWAIT" +
self.floorPortion])
                goalRoute.append("L")
                goalRoute.append(self.landmarks.dictdict[robLobbyDict][goal])
                return goalRoute

        #non lobby goals
        #on the same wing but not lobby goals
        elif goal[0] == str(self.floor) and goal[3] == self.floorPortion:
            goalRoute.append(self.landmarks.dictdict[goalDict][goal])
            return goalRoute

        #on the same floor
        elif goal[3] != self.floorPortion and goal[0] == str(self.floor):
            if self.floorPortion != "L":
                goalRoute.append(self.landmarks.dictdict[robcurrentDict]
["AWAITEXIT"])
                goalRoute.append("SLEEP")
                goalRoute.append(self.landmarks.dictdict[robLobbyDict]["AWAIT" +
self.floorPortion])
                goalRoute.append("L")
            goalRoute.append(self.landmarks.dictdict[robLobbyDict]["AWAIT" +
goal[3]])
            goalRoute.append("SLEEP")
            goalRoute.append(self.landmarks.dictdict[goalDict]["AWAITEXIT"])
            goalRoute.append(goal[3])
            goalRoute.append(self.landmarks.dictdict[goalDict][goal])
            return goalRoute

        #not on the same floor
        elif goal[0] != str(self.floor):
            if self.floorPortion != "L":
                goalRoute.append(self.landmarks.dictdict[robcurrentDict]
["AWAITEXIT"])
                goalRoute.append("SLEEP")
                f = "AWAIT" + self.floorPortion
                goalRoute.append(self.landmarks.dictdict[robLobbyDict]["AWAIT" +
self.floorPortion])
                goalRoute.append("L")
            #now get to the elevator
            goalRoute.append(self.landmarks.dictdict[robLobbyDict]["AWAITTELE"])
            goalRoute.append("ELEVATORTIME")
            goalRoute.append(self.landmarks.dictdict[robLobbyDict]["ELEVATOR"])
            goalDictLobby = "dict" + goal[0] + "L"
            goalRoute.append(self.landmarks.dictdict[goalDictLobby]
["AWAITTELE"])
            goalRoute.append(self.landmarks.dictdict[goalDictLobby]["AWAIT" +
goal[3]])
            goalRoute.append("SLEEP")
            goalRoute.append(self.landmarks.dictdict[goalDict]["AWAITEXIT"])
            goalRoute.append(goal[3])
            goalRoute.append(self.landmarks.dictdict[goalDict][goal])
            return goalRoute

    #attempt to go to a goal 3 times (so this would be one set of coordinates,
not the whole goal path)
    def attempt3(self, goal, quat, actionTime):
        errorCounter = 0
```

```
        success = False
        position = {'x': goal[0], 'y': goal[1]}
        #quaternion = {'r1': 0.000, 'r2': 0.000, 'r3': 0.000, 'r4': 1.000}
        quaternion = {'r1': quat[0], 'r2': quat[1], 'r3': quat[2], 'r4':
quat[3]}
        rospy.loginfo("Go to (%s, %s) pose", position['x'], position['y'])

        while not success and errorCounter < 3:
            success = self.nav.goto(position, quaternion, float(actionTime))
            if success:
                rospy.loginfo("GOAL REACHED")
            else:
                rospy.loginfo("GOAL FAILED")
                self.cleaner.reset_costmap_layers()

            # Sleep to give the last log messages time to be sent
            rospy.sleep(1)
            errorCounter = errorCounter + 1
        return success

    def askForHelp(self, evel):
        statement = "help, I'm on floor " + str(self.floor) + " in " +
self.floorPortion + " please rescue me :C"
        if evel:
            statement = "help, I'm stuck in the elevator please rescue me :C"
        smtp_ssl_host = 'smtp.mail.yahoo.com'
        smtp_ssl_port = 465
        username = 'definitelynotabot95@yahoo.com'
        password = 'iaintnobot'
        sender = 'definitelynotabot95@yahoo.com'
        target = 'k1630593@kcl.ac.uk'

        msg = MIMEText(statement)
        msg['Subject'] = 'HELP ' + statement
        msg['From'] = sender
        msg['To'] = target

        server = smtplib.SMTP_SSL(smtp_ssl_host, smtp_ssl_port)
        server.login(username, password)
        server.sendmail(sender, target, msg.as_string())
        server.quit()

    def debugger(self):
        print(f.generateRoute("703N"))
        print(f.generateRoute("MTOILET"))
        print(f.generateRoute("603N"))

if __name__ == "__main__":
    f = Controller()
    f.nav.makesmallspin()
    f.debugger()
    while True:
        room = raw_input("enter a room: ")
        if room[0] != str(f.floor) and len(room) == 4:
            f.goFirst(room)
            f.evelEvent()
        elif len(room) != 4 or room[0] == str(f.floor):
            f.goFirst(room)
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(simple_navigation_goals)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  actionlib
  move_base_msgs
  move_base
  geometry_msgs
  nav_msgs
  rospy
  std_msgs
  message_generation
  multi_map_server #is this how its done?
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message types you use in
##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
## * In the file package.xml:
##   * add a build_depend tag for "message_generation"
##   * add a build_depend and a exec_depend tag for each package in MSG_DEP_SET
##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
##     but can be declared for certainty nonetheless:
##     * add a exec_depend tag for "message_runtime"
## * In this file (CMakeLists.txt):
##   * add "message_generation" and every package in MSG_DEP_SET to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * add "message_runtime" and every package in MSG_DEP_SET to
##     catkin_package(CATKIN_DEPENDS ...)
##   * uncomment the add_*_files sections below as needed
##     and list every .msg/.srv/.action file to be processed
##   * uncomment the generate_messages entry below
##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
#add_service_files(
#   FILES
```

```
#   set_map_filename.srv
#   Service2.srv
#)

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
#generate_messages(
#   DEPENDENCIES
#   move_base_msgs
#   std_msgs
# )

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##     and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options(
#   cfg/DynReconf1.cfg
#   cfg/DynReconf2.cfg
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also
need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also
need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES simple_navigation_goals
#  CATKIN_DEPENDS actionlib move_base_msgs rospy
#  DEPENDS system_lib
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
```

```
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/simple_navigation_goals.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} $
{catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't
collide
# add_executable(${PROJECT_NAME}_node src/simple_navigation_goals_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames
the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg
someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX
"")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} $
{catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
install(PROGRAMS
  src/controller_framework_node.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

## Mark executables and/or libraries for installation
# install(TARGETS ${PROJECT_NAME} ${PROJECT_NAME}_node
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
```

```
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_simple_navigation_goals.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>simple_navigation_goals</name>
  <version>0.0.0</version>
  <description>The simple_navigation_goals package</description>

  <maintainer email="shuistlo@gmail.com">shu</maintainer>
  <license>BSD</license>

  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec
dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <build_depend>message_generation</build_depend>
  <!-- Use build_export_depend for packages you need in order to build against
this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use exec_depend for packages you need at runtime: -->
  <exec_depend>message_runtime</exec_depend>
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <!-- Use doc_depend for packages you need only for building documentation: -->
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>actionlib</build_depend>
  <build_depend>move_base_msgs</build_depend>
  <build_depend>rospy</build_depend>
  <build_export_depend>actionlib</build_export_depend>
  <build_export_depend>move_base_msgs</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
```

```
  <build_export_depend>multi_map_server</build_export_depend>
  <exec_depend>actionlib</exec_depend>
  <exec_depend>move_base_msgs</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>multi_map_server</exec_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

setup.py

```
#!/usr/bin/env python

## ! DO NOT MANUALLY INVOKE THIS setup.py, USE CATKIN INSTEAD

from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

# fetch values from package.xml
setup_args = generate_distutils_setup(
    packages=['controller_framework'],
    scripts=[],
    package_dir={'': 'src'}
)

setup(**setup_args)
```

main.cpp

```
/*
 * Copyright (c) 2008, Willow Garage, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the Willow Garage, Inc. nor the names of its
 *       contributors may be used to endorse or promote products derived from
 *       this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
```

```
/* Author: Brian Gerkey */

#define USAGE "\nUSAGE: multi_map_server <map.yaml>\n" \
              "  map.yaml: map description file\n" \
              "DEPRECATED USAGE: multi_map_server <map> <resolution>\n" \
              "  map: image file to load\n"\
              "  resolution: map resolution [meters/pixel]"

#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#include <fstream>

#include "ros/ros.h"
#include "ros/console.h"
#include "multi_map_server/image_loader.h"
#include "multi_map_server/set_map_filename.h"
#include "nav_msgs/MapMetaData.h"
#include "yaml-cpp/yaml.h"

#ifdef HAVE_YAMLCPP_GT_0_5_0
// The >> operator disappeared in yaml-cpp 0.5, so this function is
// added to provide support for code written under the yaml-cpp 0.3 API.
template<typename T>
void operator >> (const YAML::Node& node, T& i)
{
  i = node.as<T>();
}
#endif

class MapServer
{
  public:
    /** Trivial constructor */
    MapServer(const std::string& fname, double res)
    {
      std::string mapfname = "";
      double origin[3];
      int negate;
      double occ_th, free_th;
      MapMode mode = TRINARY;
      std::string frame_id;
      ros::NodeHandle private_nh("~");
      private_nh.param("frame_id", frame_id, std::string("map"));
      deprecated = (res != 0);
      if (!deprecated) {
        //mapfname = fname + ".pgm";
        //std::ifstream fin((fname + ".yaml").c_str());
        std::ifstream fin(fname.c_str());
        if (fin.fail()) {
          ROS_ERROR("Multi_map_server could not open %s.", fname.c_str());
          exit(-1);
        }
#ifdef HAVE_YAMLCPP_GT_0_5_0
        // The document loading process changed in yaml-cpp 0.5.
        YAML::Node doc = YAML::Load(fin);
#else
        YAML::Parser parser(fin);
        YAML::Node doc;
        parser.GetNextDocument(doc);
#endif
        try {
          doc["resolution"] >> res;
```

```
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain a resolution tag or it is
invalid.");
          exit(-1);
        }
        try {
          doc["negate"] >> negate;
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain a negate tag or it is invalid.");
          exit(-1);
        }
        try {
          doc["occupied_thresh"] >> occ_th;
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain an occupied_thresh tag or it is
invalid.");
          exit(-1);
        }
        try {
          doc["free_thresh"] >> free_th;
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain a free_thresh tag or it is
invalid.");
          exit(-1);
        }
        try {
          std::string modeS = "";
          doc["mode"] >> modeS;

          if(modeS=="trinary")
            mode = TRINARY;
          else if(modeS=="scale")
            mode = SCALE;
          else if(modeS=="raw")
            mode = RAW;
          else{
            ROS_ERROR("Invalid mode tag \"%s\".", modeS.c_str());
            exit(-1);
          }
        } catch (YAML::Exception) {
          ROS_DEBUG("The map does not contain a mode tag or it is invalid...
assuming Trinary");
          mode = TRINARY;
        }
        try {
          doc["origin"][0] >> origin[0];
          doc["origin"][1] >> origin[1];
          doc["origin"][2] >> origin[2];
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain an origin tag or it is invalid.");
          exit(-1);
        }
        try {
          doc["image"] >> mapfname;
          // TODO: make this path-handling more robust
          if(mapfname.size() == 0)
          {
            ROS_ERROR("The image tag cannot be an empty string.");
            exit(-1);
          }
          if(mapfname[0] != '/')
          {
            // dirname can modify what you pass it
            char* fname_copy = strdup(fname.c_str());
```

```
            mapfname = std::string(dirname(fname_copy)) + '/' + mapfname;
            free(fname_copy);
          }
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain an image tag or it is invalid.");
          exit(-1);
        }
      } else {
        private_nh.param("negate", negate, 0);
        private_nh.param("occupied_thresh", occ_th, 0.65);
        private_nh.param("free_thresh", free_th, 0.196);
        mapfname = fname;
        origin[0] = origin[1] = origin[2] = 0.0;
      }

      ROS_INFO("Loading map from image \"%s\"", mapfname.c_str());
      try
      {

multi_map_server::loadMapFromFile(&map_resp_,mapfname.c_str(),res,negate,occ_th,
free_th, origin, mode);
      }
      catch (std::runtime_error e)
      {
          ROS_ERROR("%s", e.what());
          exit(-1);
      }
      // To make sure get a consistent time in simulation
      ros::Time::waitForValid();
      map_resp_.map.info.map_load_time = ros::Time::now();
      map_resp_.map.header.frame_id = frame_id;
      map_resp_.map.header.stamp = ros::Time::now();
      ROS_INFO("Read a %d X %d map @ %.3lf m/cell",
               map_resp_.map.info.width,
               map_resp_.map.info.height,
               map_resp_.map.info.resolution);
      meta_data_message_ = map_resp_.map.info;

      static_map_service = n.advertiseService("static_map",
&MapServer::mapCallback, this);
      //pub = n.advertise<nav_msgs::MapMetaData>("map_metadata", 1,
        set_map_filename_service = n.advertiseService("set_map_filename",
&MapServer::set_map_filename_callback, this);

      // Latched publisher for metadata
      metadata_pub= n.advertise<nav_msgs::MapMetaData>("map_metadata", 1, true);
      metadata_pub.publish( meta_data_message_ );

      // Latched publisher for data
      map_pub = n.advertise<nav_msgs::OccupancyGrid>("map", 1, true);
      map_pub.publish( map_resp_.map );
    }

  private:
    ros::NodeHandle n;
    ros::Publisher map_pub;
    ros::Publisher metadata_pub;
    ros::ServiceServer static_map_service, set_map_filename_service;
    bool deprecated;

    /** Callback invoked when someone requests our service */
    bool mapCallback(nav_msgs::GetMap::Request  &req,
                     nav_msgs::GetMap::Response &res )
    {
```

```
      // request is empty; we ignore it

      // = operator is overloaded to make deep copy (tricky!)
      res = map_resp_;
      ROS_INFO("Sending map");

      return true;
    }

      /** THIS IS THE CALLBACK FOR MY SERVICE set_map_filename **/
      bool set_map_filename_callback(multi_map_server::set_map_filename::Request
&req, multi_map_server::set_map_filename::Request &res)
      {
        double resolution;

      std::string mapfname = "";
      double origin[3];
      int negate;
      double occ_th, free_th;
      MapMode mode = TRINARY;
      std::string frame_id;
      ros::NodeHandle private_nh("~");
      private_nh.param("frame_id", frame_id, std::string("map"));
        std::ifstream fin(req.fileName.c_str());

#ifdef HAVE_YAMLCPP_GT_0_5_0
        // The document loading process changed in yaml-cpp 0.5.
        YAML::Node doc = YAML::Load(fin);
#else
        YAML::Parser parser(fin);
        YAML::Node doc;
        parser.GetNextDocument(doc);
#endif
        try {
          doc["resolution"] >> resolution;
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain a resolution tag or it is
invalid.");
              return false;
        }
        try {
          doc["negate"] >> negate;
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain a negate tag or it is invalid.");
          return false;
        }
        try {
          doc["occupied_thresh"] >> occ_th;
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain an occupied_thresh tag or it is
invalid.");
          return false;
        }
        try {
          doc["free_thresh"] >> free_th;
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain a free_thresh tag or it is
invalid.");
          return false;
        }
        try {
          std::string modeS = "";
          doc["mode"] >> modeS;
```

```
          if(modeS=="trinary")
            mode = TRINARY;
          else if(modeS=="scale")
            mode = SCALE;
          else if(modeS=="raw")
            mode = RAW;
          else{
            ROS_ERROR("Invalid mode tag \"%s\".", modeS.c_str());
            return false;
          }
        } catch (YAML::Exception) {
          ROS_DEBUG("The map does not contain a mode tag or it is invalid...
assuming Trinary");
          mode = TRINARY;
        }
        try {
          doc["origin"][0] >> origin[0];
          doc["origin"][1] >> origin[1];
          doc["origin"][2] >> origin[2];
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain an origin tag or it is invalid.");
          return false;
        }
        try {
          doc["image"] >> mapfname;
          if(mapfname.size() == 0)
          {
            ROS_ERROR("The image tag cannot be an empty string.");
            return false;
          }
          if(mapfname[0] != '/')
          {
            // dirname can modify what you pass it
            char* fname_copy = strdup(req.fileName.c_str());
            mapfname = std::string(dirname(fname_copy)) + '/' + mapfname;
            free(fname_copy);
          }
        } catch (YAML::InvalidScalar) {
          ROS_ERROR("The map does not contain an image tag or it is invalid.");
          return false;
        }
      private_nh.param("negate", negate, 0);
      private_nh.param("occupied_thresh", occ_th, 0.65);
      private_nh.param("free_thresh", free_th, 0.196);
      origin[0] = origin[1] = origin[2] = 0.0;

      ROS_INFO("Loading map from image \"%s\"", mapfname.c_str());
      try
      {

multi_map_server::loadMapFromFile(&map_resp_,mapfname.c_str(),resolution,negate,
occ_th,free_th, origin, mode);
      }
      catch (std::runtime_error e)
      {
          ROS_ERROR("%s", e.what());
          return false;
      }
      // To make sure get a consistent time in simulation
      ros::Time::waitForValid();
      map_resp_.map.info.map_load_time = ros::Time::now();
      map_resp_.map.header.frame_id = frame_id;
      map_resp_.map.header.stamp = ros::Time::now();
      ROS_INFO("Read a %d X %d map @ %.3lf m/cell",
```

```
                map_resp_.map.info.width,
                map_resp_.map.info.height,
                map_resp_.map.info.resolution);
      meta_data_message_ = map_resp_.map.info;

      metadata_pub.publish( meta_data_message_ );
      map_pub.publish( map_resp_.map );

      return true;
      }

    /** The map data is cached here, to be sent out to service callers
     */
    nav_msgs::MapMetaData meta_data_message_;
    nav_msgs::GetMap::Response map_resp_;

    /*
    void metadataSubscriptionCallback(const ros::SingleSubscriberPublisher& pub)
    {
      pub.publish( meta_data_message_ );
    }
    */

};

int main(int argc, char **argv)
{
  ros::init(argc, argv, "multi_map_server", ros::init_options::AnonymousName);
  if(argc != 3 && argc != 2)
  {
    ROS_ERROR("%s", USAGE);
    exit(-1);
  }
  if (argc != 2) {
    ROS_WARN("Using deprecated map server interface. Please switch to new
interface.");
  }
  std::string fname(argv[1]);
  double res = (argc == 2) ? 0.0 : atof(argv[2]);

  try
  {
    MapServer ms(fname, res);
    ros::spin();
  }
  catch(std::runtime_error& e)
  {
    ROS_ERROR("multi_map_server exception: %s", e.what());
    return -1;
  }

  return 0;
}
```

set_map_filename.srv

```
# Set a new map
string fileName
---
bool success
```

5thedited.world

```
include "/opt/ros/kinetic/share/turtlebot_stage/maps/stage/turtlebot.inc"

define floorplan model
(
  # sombre, sensible, artistic
  color "gray30"

  # most maps will need a bounding box
  boundary 1

  gui_nose 0
  gui_grid 0
  gui_outline 0
  gripper_return 0
  fiducial_return 0
  laser_return 1
)

resolution 0.05
interval_sim 100  # simulation timestep in milliseconds

window
(
  size [ 600.0 700.0 ]
  center [ 0.0 0.0 ]
  rotate [ 0.0 0.0 ]
  scale 60
)

floorplan
(
  name "5thedited.world"
  bitmap "5thedited.png"
  size [ 76.0 28.95 2.0 ]
  pose [  38.0  14.475 0.0 0.0 ]
)

# throw in a robot
turtlebot
(
  pose [ 38.0 17.0 0.0 0.0 ]
  name "turtlebot"
  color "black"
)
```

5thedited.yaml

```
image: /home/shu/maps/5thedited.pgm
resolution: 0.050000
origin: [-1,-1, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

6thedited.world

```
include "/opt/ros/kinetic/share/turtlebot_stage/maps/stage/turtlebot.inc"

define floorplan model
(
  # sombre, sensible, artistic
  color "gray30"

  # most maps will need a bounding box
  boundary 1

  gui_nose 0
  gui_grid 0
  gui_outline 0
  gripper_return 0
  fiducial_return 0
  laser_return 1
)

resolution 0.05
interval_sim 100  # simulation timestep in milliseconds

window
(
  size [ 600.0 700.0 ]
  center [ 0.0 0.0 ]
  rotate [ 0.0 0.0 ]
  scale 60
)

floorplan
(
  name "6thedited.world"
  bitmap "6thedited.png"
  size [ 73.6 25.9 2.0 ]
  pose [  36.8  12.95 0.0 0.0 ]
)

# throw in a robot
turtlebot
(
  pose [ 36.5 17.0 0.0 0.0 ]
  name "turtlebot"
  color "black"
)
```

6thedited.yaml

```
image: /home/shu/maps/6thedited.pgm
resolution: 0.050000
origin: [-1,-1, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

7thedited.world

```
include "/opt/ros/kinetic/share/turtlebot_stage/maps/stage/turtlebot.inc"

define floorplan model
(
  # sombre, sensible, artistic
  color "gray30"
```

```
  # most maps will need a bounding box
  boundary 1

  gui_nose 0
  gui_grid 0
  gui_outline 0
  gripper_return 0
  fiducial_return 0
  laser_return 1
)

resolution 0.05
interval_sim 100  # simulation timestep in milliseconds

window
(
  size [ 600.0 700.0 ]
  center [ 0.0 0.0 ]
  rotate [ 0.0 0.0 ]
  scale 60
)

floorplan
(
  name "7thedited.world"
  bitmap "7thedited.png"
  size [ 73.65 26.1 2.0 ]
  pose [  36.825  13.05 0.0 0.0 ]
)

# throw in a robot
turtlebot
(
  pose [ 36.5 17.0 0.0 0.0 ]
  name "turtlebot"
  color "black"
)
```

7thedited.yaml

```
image: /home/shu/maps/7thedited.pgm
resolution: 0.050000
origin: [-1,-1, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```