Westfälische Hochschule

University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Dokumentation

Projekt ARO

Aufgabenstellung:

Challenge: Autonom um die Hochschule fahren mit dem Summit XL

Autoren:

Marius Grevelhörster	201326429
Tim Kestermann	201328224
Martin Kondring	201424335
Eric Dapper	201220826

Studiengang:

Master Maschinenbau

Fachbereich:

Maschinenbau

Betreuer:

Prof. Dr. Olaf Just

Prof. Dr. Martin Maß

Abgabedatum:

06.02.2019

Inhaltsverzeichnis

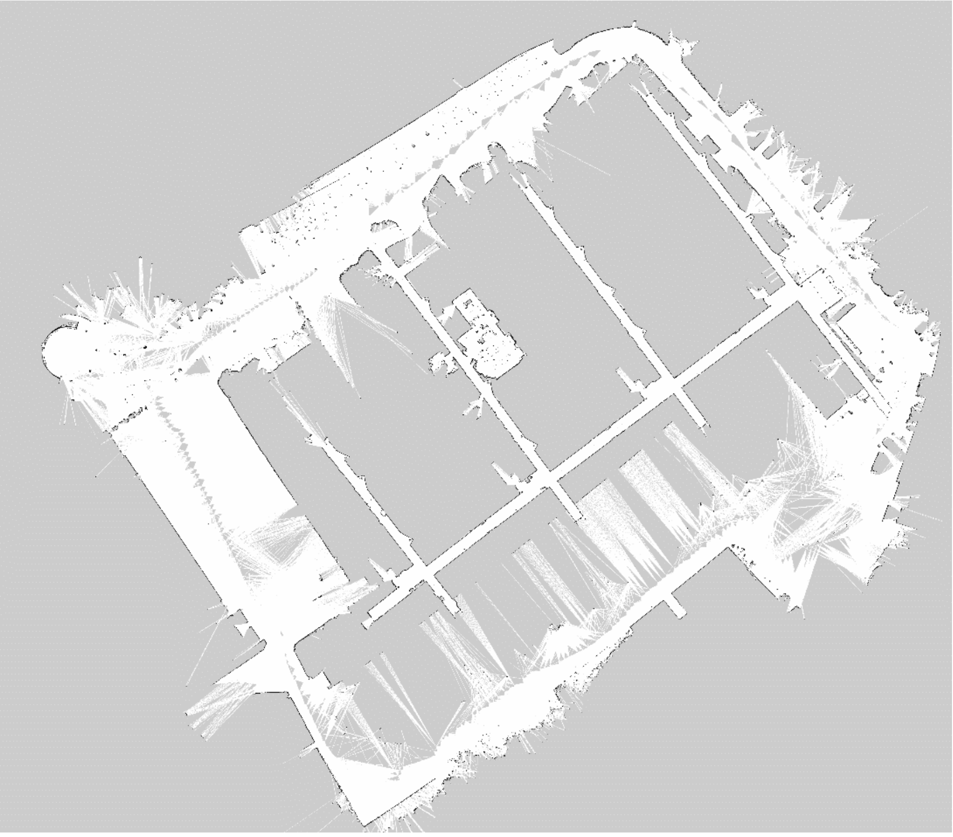
1 Durchführung

Der folgende Bericht umfasst ergänzende Unterlagen zur Präsentation vom 06.02.2019.

1.1 Gimp

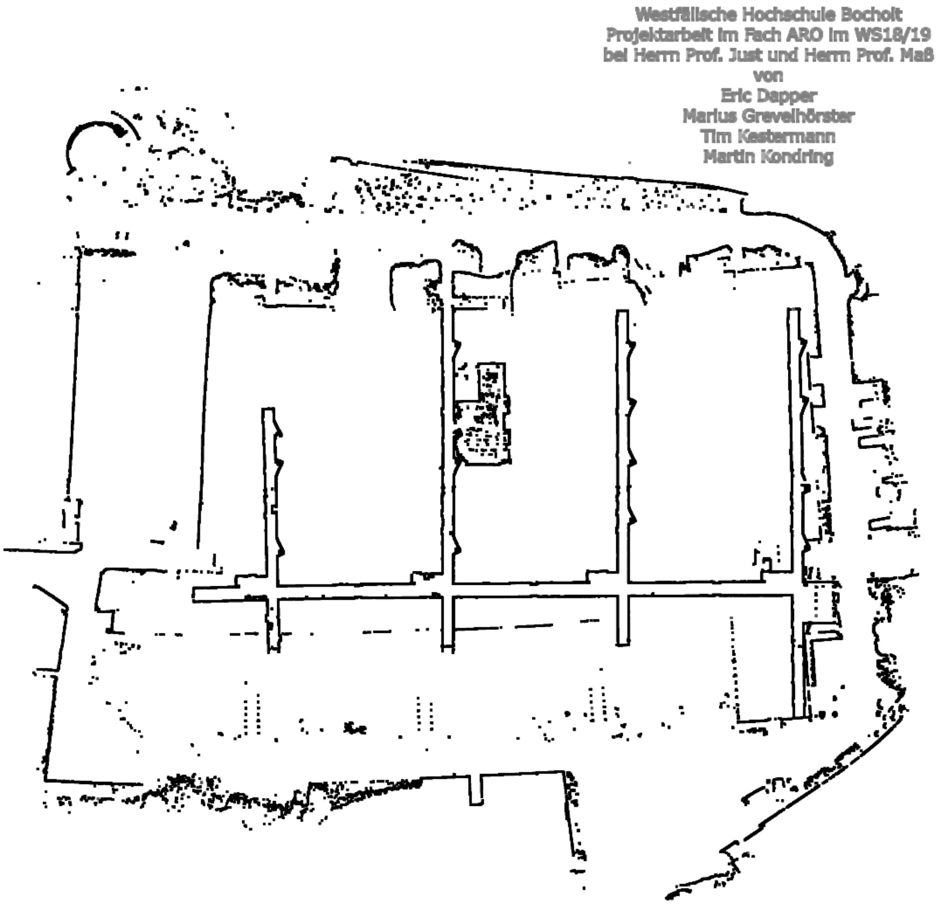
Mit Gimp wird die .pgm-Datei aufgearbeitet für die Lokalisations-Map als .pgm exportiert. Für die Simulationsumgebung muss für die weitere Aufarbeitung das Modell als .svg-Datei vorliegen.

Anmerkung: Wichtig hierbei ist das das Bild nicht beschnitten wird, um Skalierungsfehler von vornerein zu verhindern. Die Kanten/Punkte müssen dick genug sein um das Verfahren und wiederfinden des realen und simulierten Roboters zu ermöglichen.

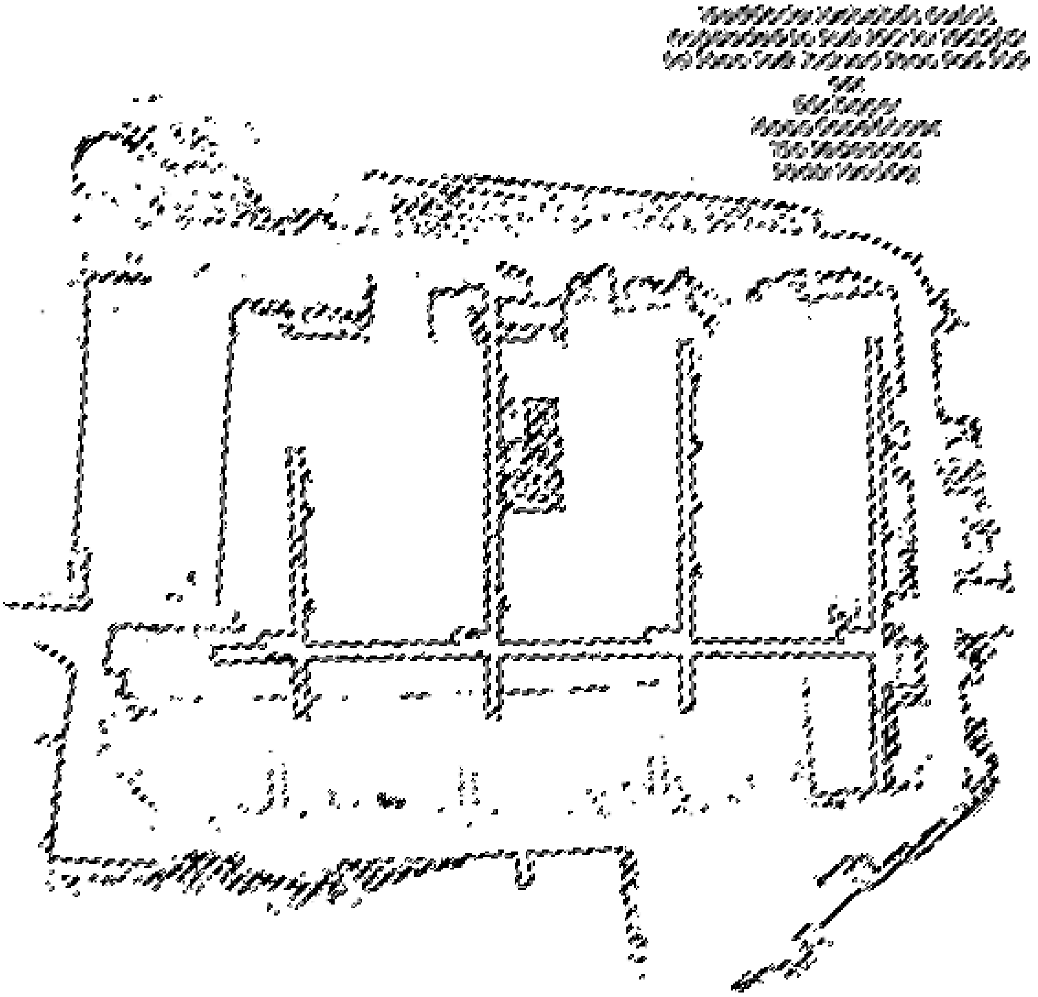


Mit verschiedenen Filtern, Kantenglättung und Schwellwert die Map aufarbeiten.

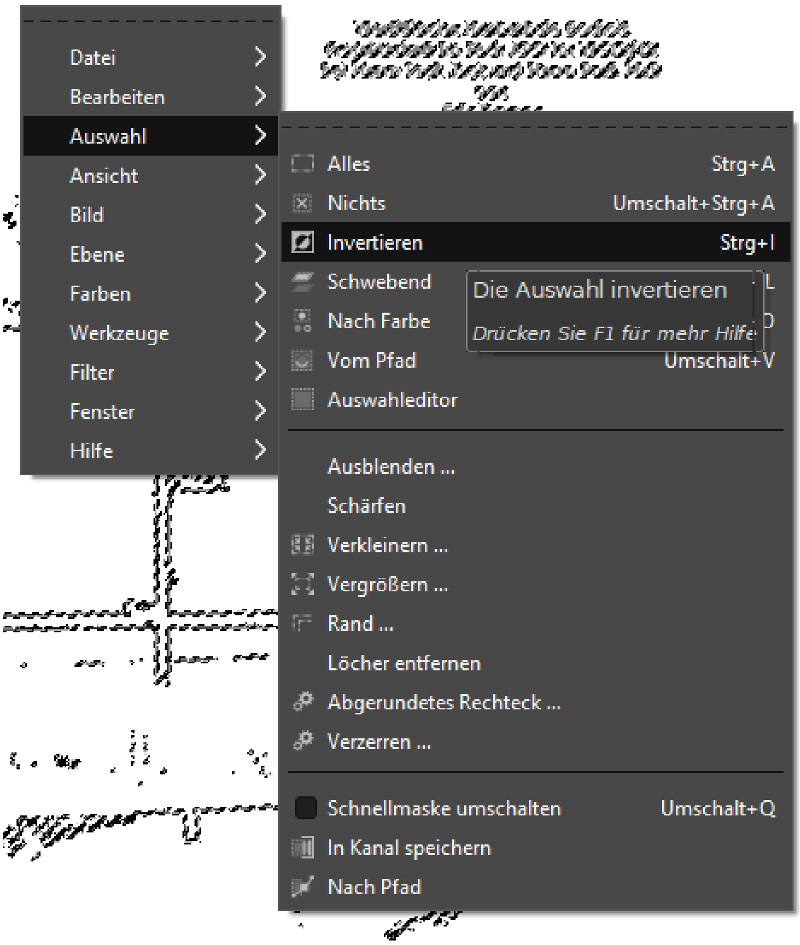
Durchführung


Westfälische Hochschule Bocholt
Projektarbeit im Fach ARO im WS18/19
bei Herrn Prof. Just und Herrn Prof. Maß
von
Eric Dapper
Marius Grevelhörster
Tim Kestermann
Martin Kondring

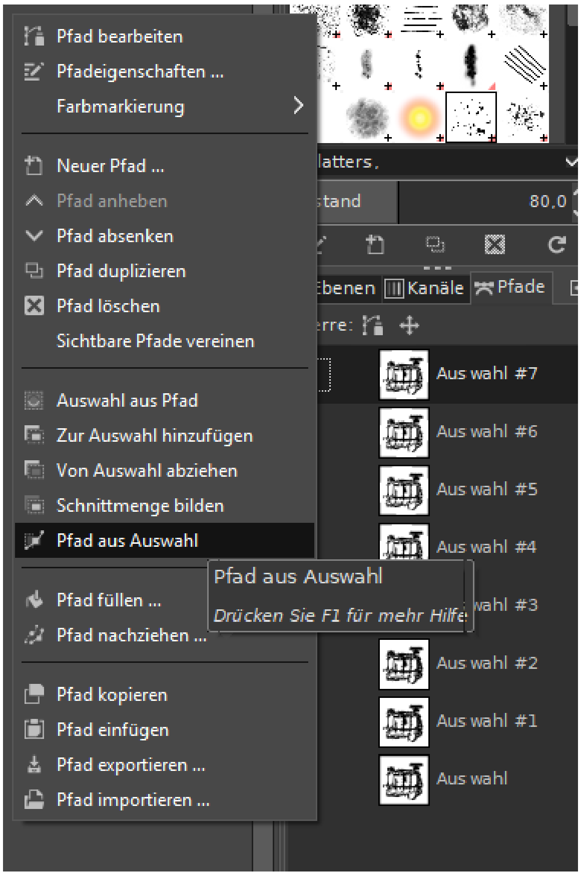
Mit „Nach Farbe auswählen“ den weißen Bereich auswählen



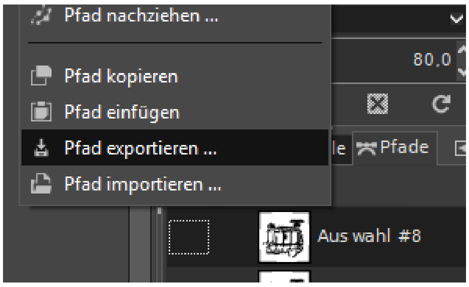
Auswahl invertieren



Über „Pfad aus Auswahl“ Pfad erzeugen.



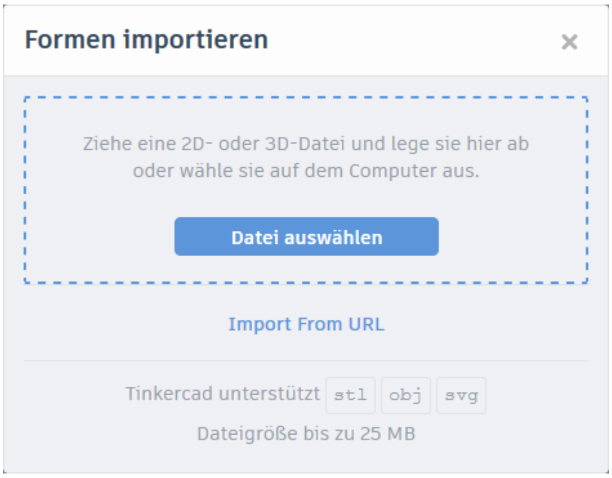
Pfad exportieren und als .svg-Datei speichern.



1.2 Tinkercad

Tinkercad dient dazu aus der .svg-Datei eine .stl-Datei zu erzeugen.

https://www.tinkercad.com/things/



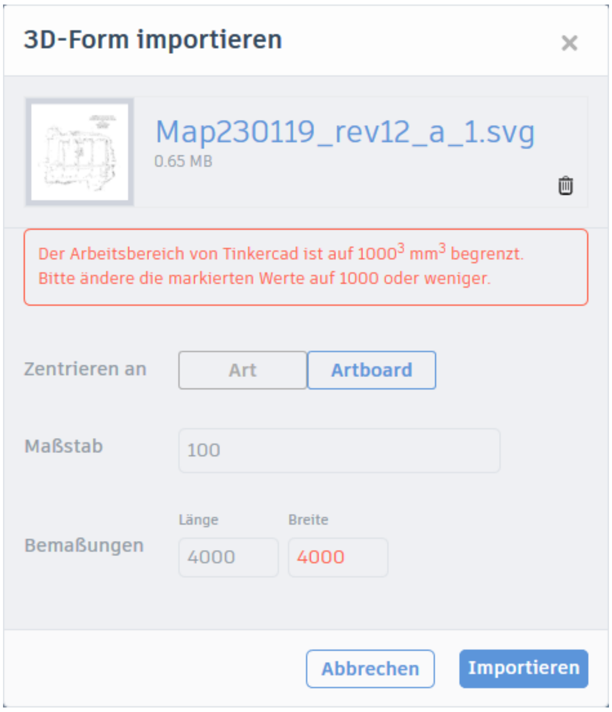
Formen importieren

Ziehe eine 2D- oder 3D-Datei und lege sie hier ab oder wähle sie auf dem Computer aus.

Tinkercad unterstützt stl obj svg

Dateigröße bis zu 25 MB

Maßstab beim importieren anpassen



3D-Form importieren

Map230119_rev12_a_1.svg

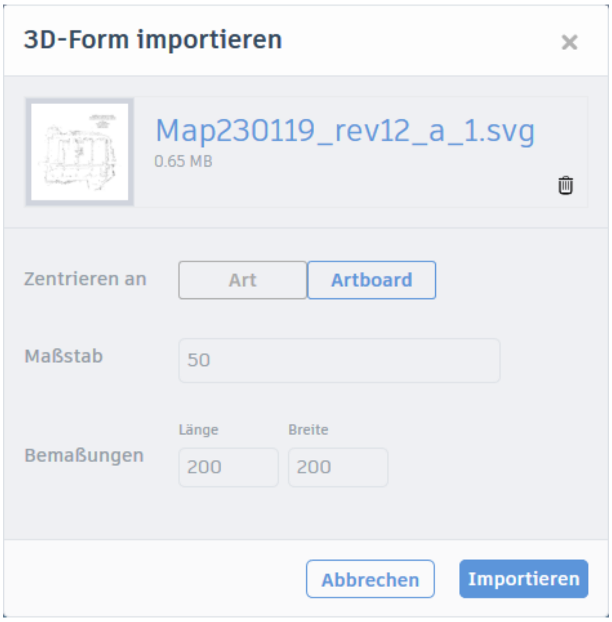
0.65 MB

Der Arbeitsbereich von Tinkercad ist auf 1000^3 mm^3 begrenzt. Bitte ändere die markierten Werte auf 1000 oder weniger.

Zentrieren an: Art [✓ Artboard]

Maßstab: 100

Bemaßungen: Länge 4000, Breite 4000



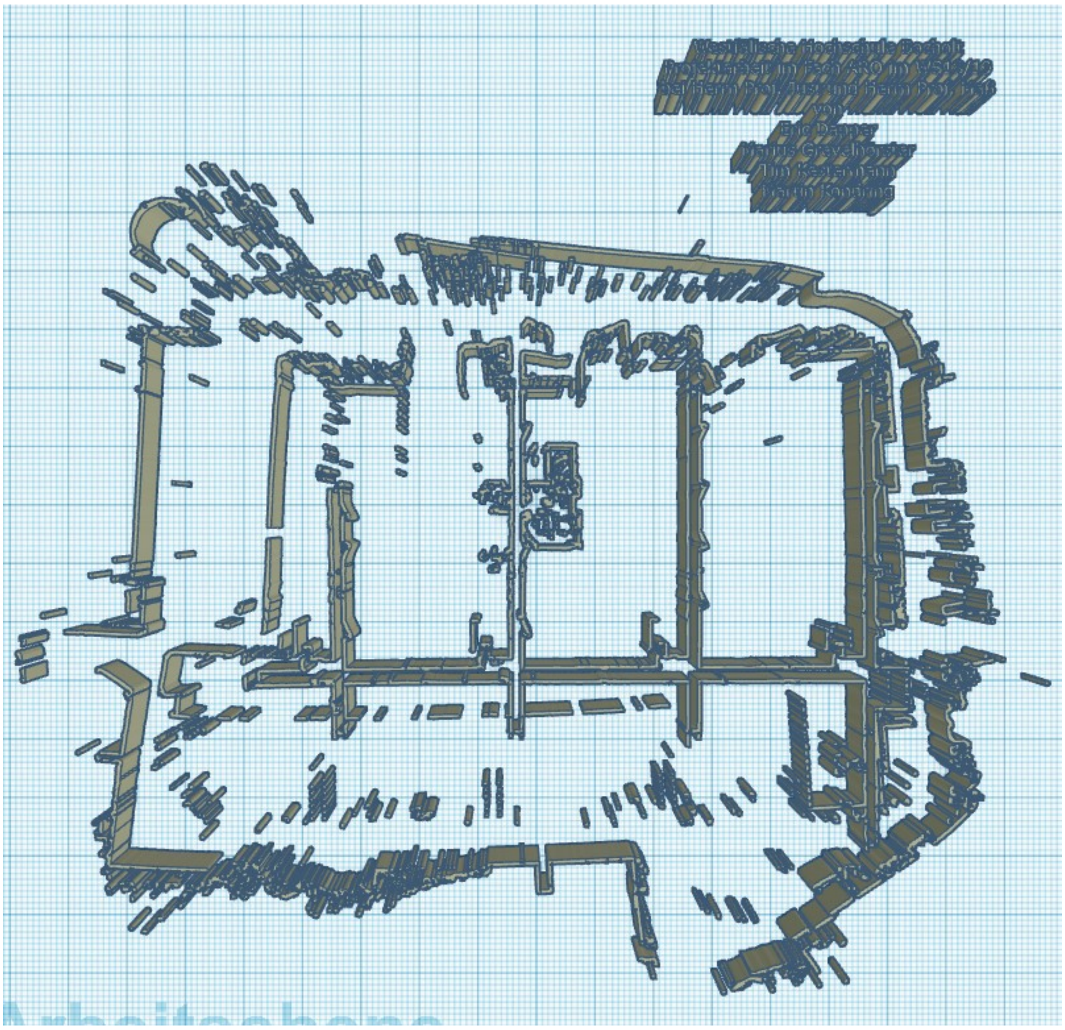
3D-Form importieren

Map230119_rev12_a_1.svg
0.65 MB

Zentrieren an: Art / Artboard

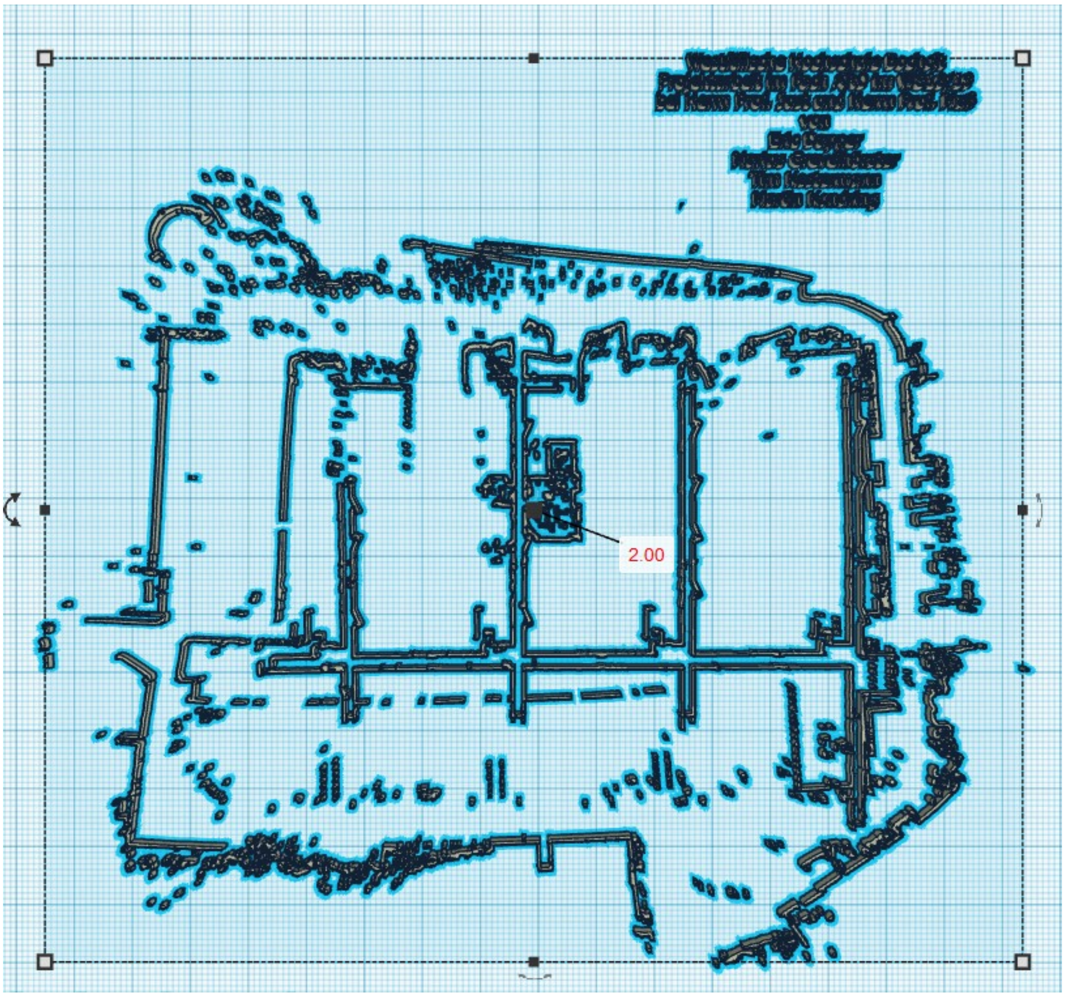
Maßstab: 50

Bemaßungen: Länge 200, Breite 200

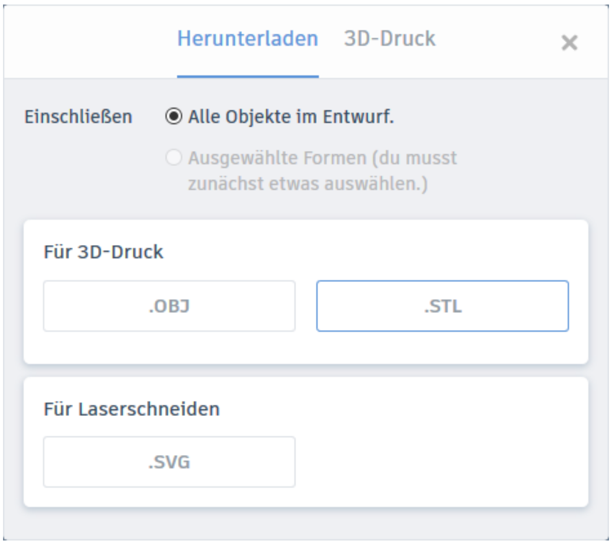


Westfälische Hochschule Bocholt
Projektarbeit im Fach ARO im WS18/19
bei Herrn Prof. Just und Herrn Prof.
von
Tim Kestermann

Höhe auf 2,0 mm anpassen



Als .stl-Datei exportieren



Herunterladen 3D-Druck ×

Einschließen

● Alle Objekte im Entwurf.

○ Ausgewählte Formen (du musst zunächst etwas auswählen.)

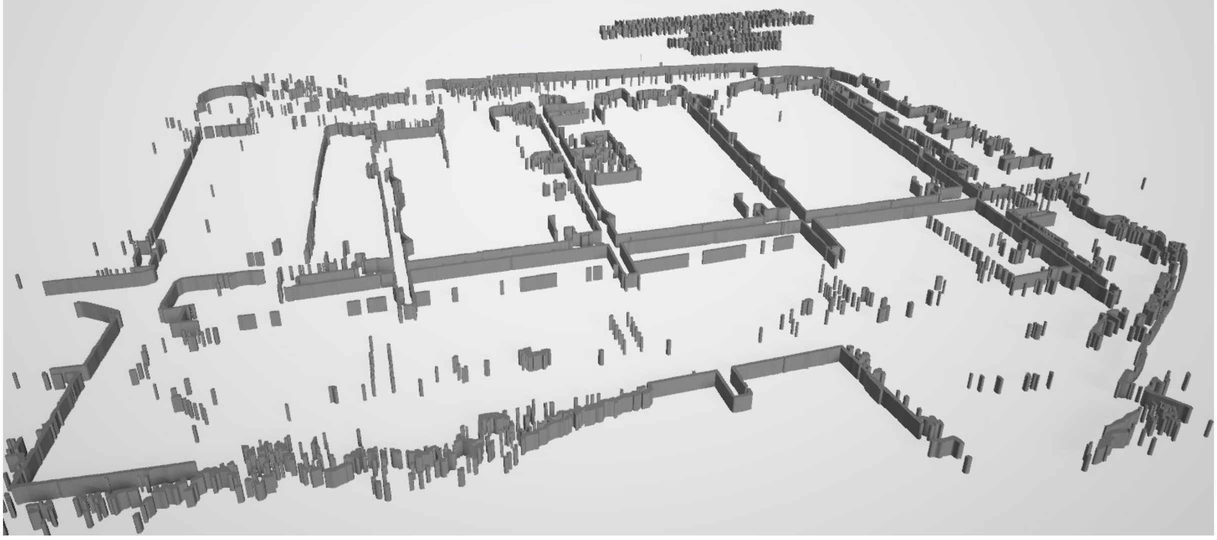
Für 3D-Druck

.OBJ .STL

Für Laserschneiden

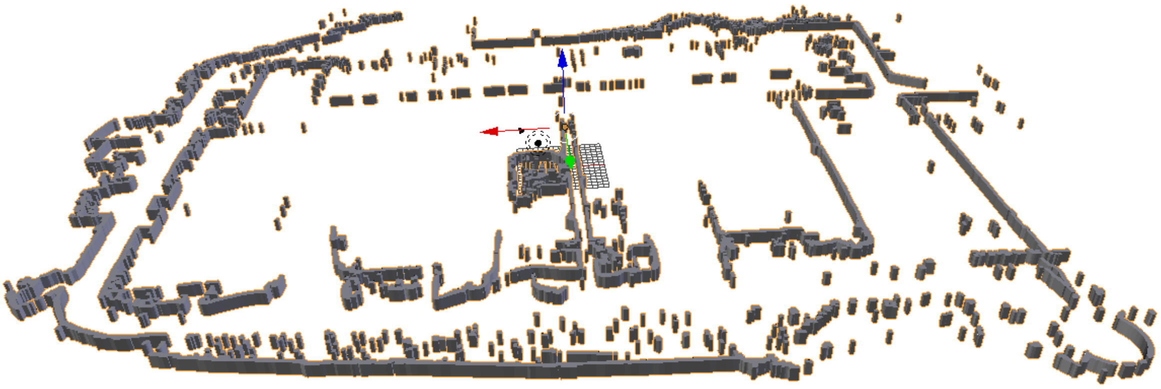
.SVG

Fertig exportierte .stl-Modell



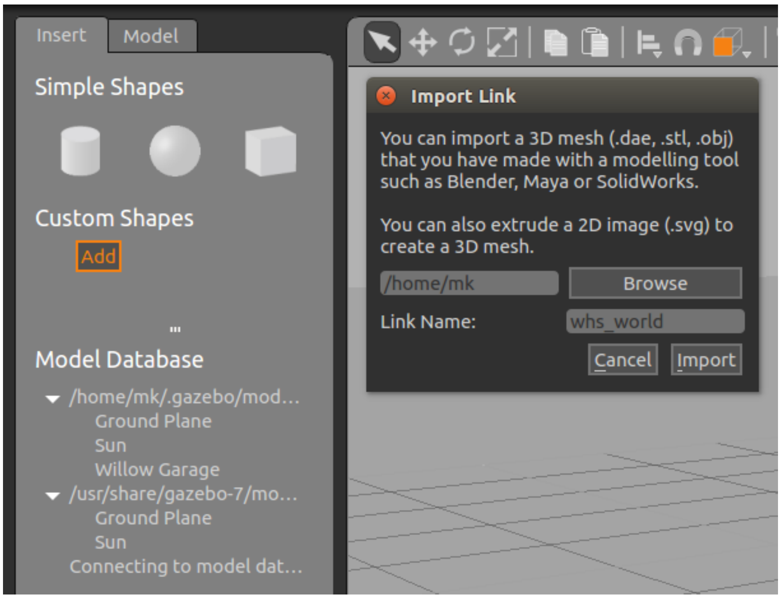
1.3 Blender

Mittels Blender (https://www.blender.org/) das .stl-Dateiformat in .dae-Dateiformat ändern.



1.4 Gazebo

- Gazebo öffnen
- Model-Editor öffnen (Strg + M)

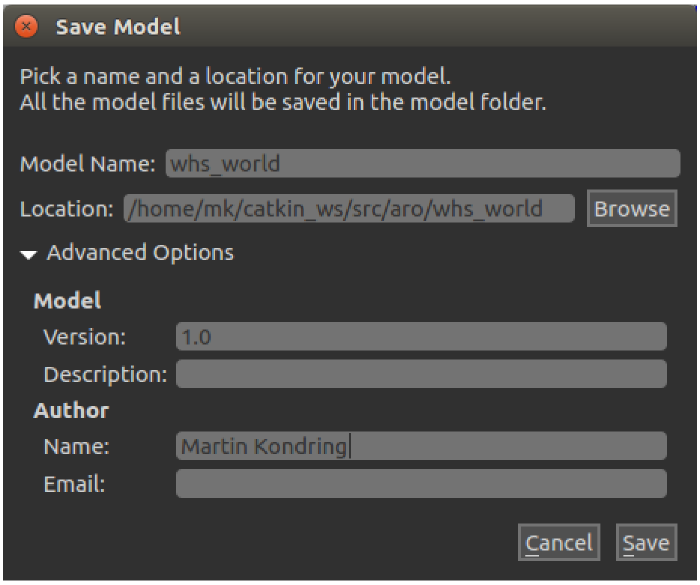


You can import a 3D mesh (.dae, .stl, .obj) that you have made with a modelling tool such as Blender, Maya or SolidWorks.

You can also extrude a 2D image (.svg) to create a 3D mesh.

/home/mk

Link Name: whs_world



Pick a name and a location for your model.
All the model files will be saved in the model folder.

Model Name: whs_world

Location: /home/mk/catkin_ws/src/aro/whs_world

Model

Version: 1.0

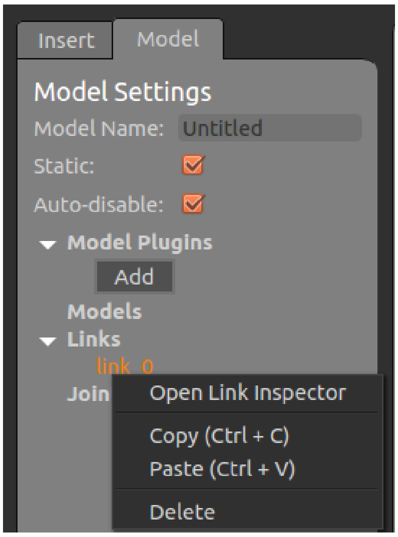
Description:

Author

Name: Martin Kondring

Email:

Link anlegen und bearbeiten

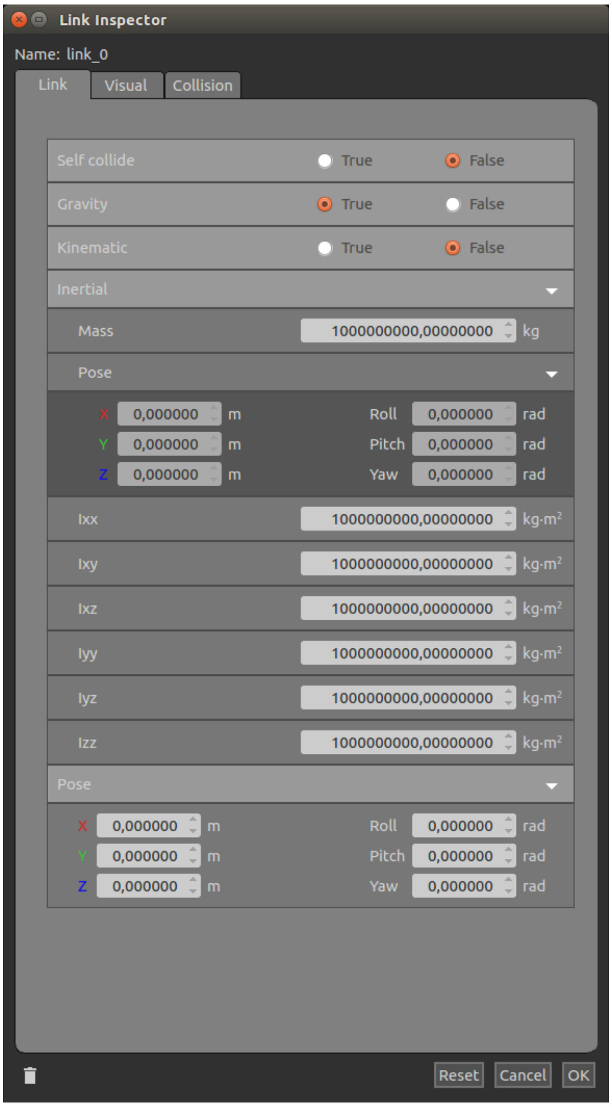


Model Name: Untitled

Static: ☑

Auto-disable: ☑

Parameter anpassen



Name: link_0

Self collide ○ True ● False

Gravity ● True ○ False

Kinematic ○ True ● False

Inertial

Mass 1000000000,00000000 kg

Pose

X 0,000000 m Roll 0,000000 rad

Y 0,000000 m Pitch 0,000000 rad

Z 0,000000 m Yaw 0,000000 rad

Ixx 1000000000,00000000 kg·m²

Ixy 1000000000,00000000 kg·m²

Ixz 1000000000,00000000 kg·m²

Iyy 1000000000,00000000 kg·m²

Iyz 1000000000,00000000 kg·m²

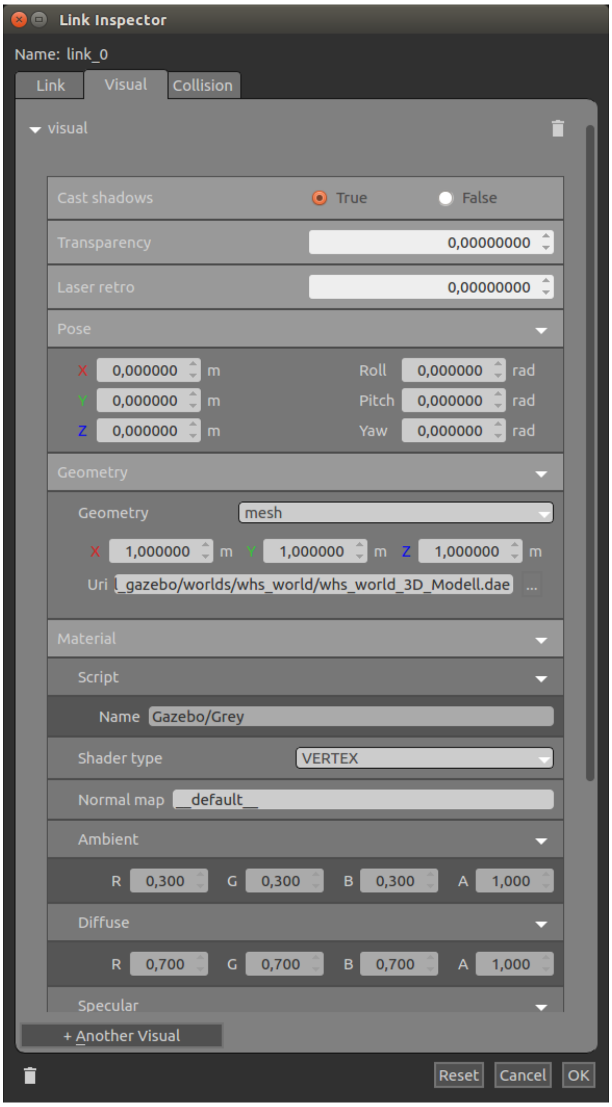
Izz 1000000000,00000000 kg·m²

Pose

X 0,000000 m Roll 0,000000 rad

Y 0,000000 m Pitch 0,000000 rad

Z 0,000000 m Yaw 0,000000 rad



Name: link_0

visual

Cast shadows ● True ○ False

Transparency 0,00000000

Laser retro 0,00000000

Pose

X 0,000000 m Roll 0,000000 rad

Y 0,000000 m Pitch 0,000000 rad

Z 0,000000 m Yaw 0,000000 rad

Geometry

Geometry mesh

X 1,000000 m Y 1,000000 m Z 1,000000 m

Uri _gazebo/worlds/whs_world/whs_world_3D_Modell.dae

Material

Script

Name Gazebo/Grey

Shader type VERTEX

Normal map __default__

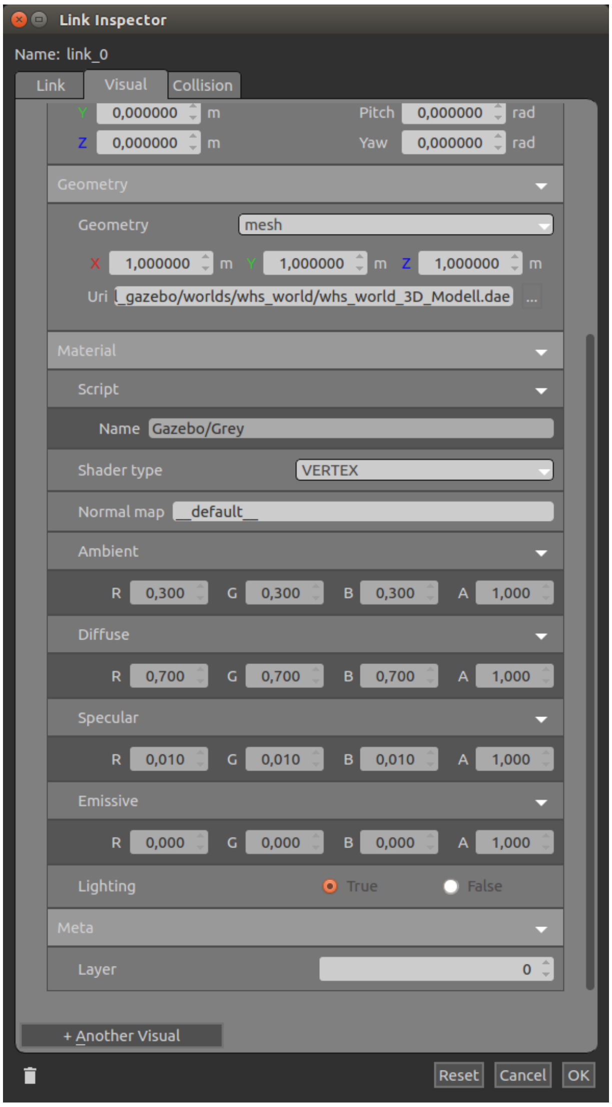
Ambient

R 0,300 G 0,300 B 0,300 A 1,000

Diffuse

R 0,700 G 0,700 B 0,700 A 1,000

Specular



Link Inspector

Name: link_0

Y 0,000000 m Pitch 0,000000 rad

Z 0,000000 m Yaw 0,000000 rad

Geometry

Geometry mesh

X 1,000000 m Y 1,000000 m Z 1,000000 m

Uri _gazebo/worlds/whs_world/whs_world_3D_Modell.dae

Material

Script

Name Gazebo/Grey

Shader type VERTEX

Normal map __default__

Ambient

R 0,300 G 0,300 B 0,300 A 1,000

Diffuse

R 0,700 G 0,700 B 0,700 A 1,000

Specular

R 0,010 G 0,010 B 0,010 A 1,000

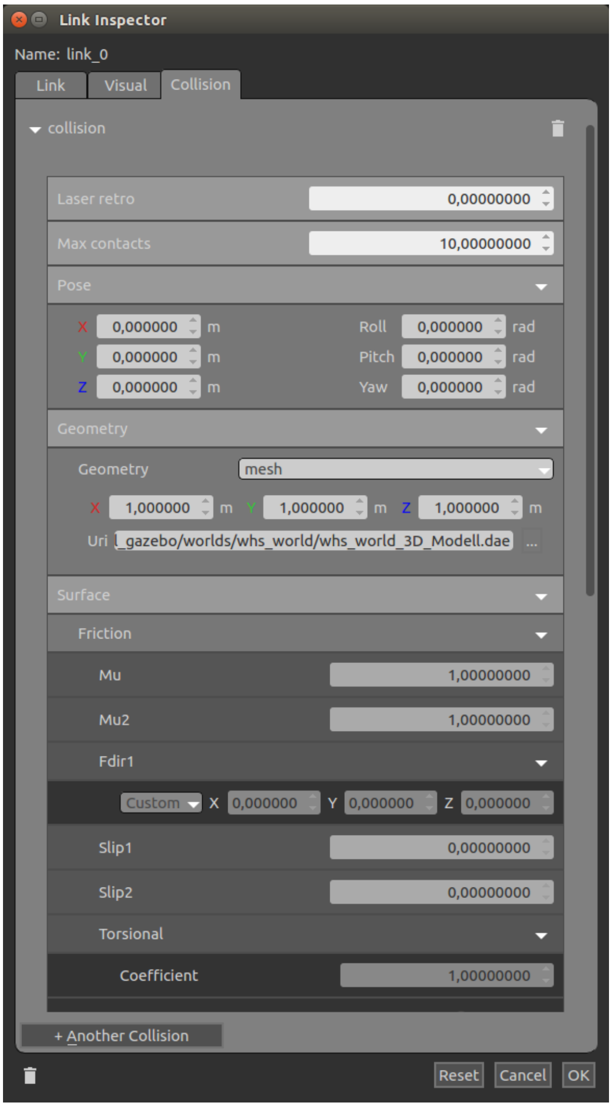
Emissive

R 0,000 G 0,000 B 0,000 A 1,000

Lighting ● True ○ False

Meta

Layer 0



Name: link_0

Laser retro 0,00000000

Max contacts 10,00000000

Pose

X 0,000000 m Roll 0,000000 rad

Y 0,000000 m Pitch 0,000000 rad

Z 0,000000 m Yaw 0,000000 rad

Geometry

Geometry mesh

X 1,000000 m Y 1,000000 m Z 1,000000 m

Uri _gazebo/worlds/whs_world/whs_world_3D_Modell.dae ...

Surface

Friction

Mu 1,00000000

Mu2 1,00000000

Fdir1

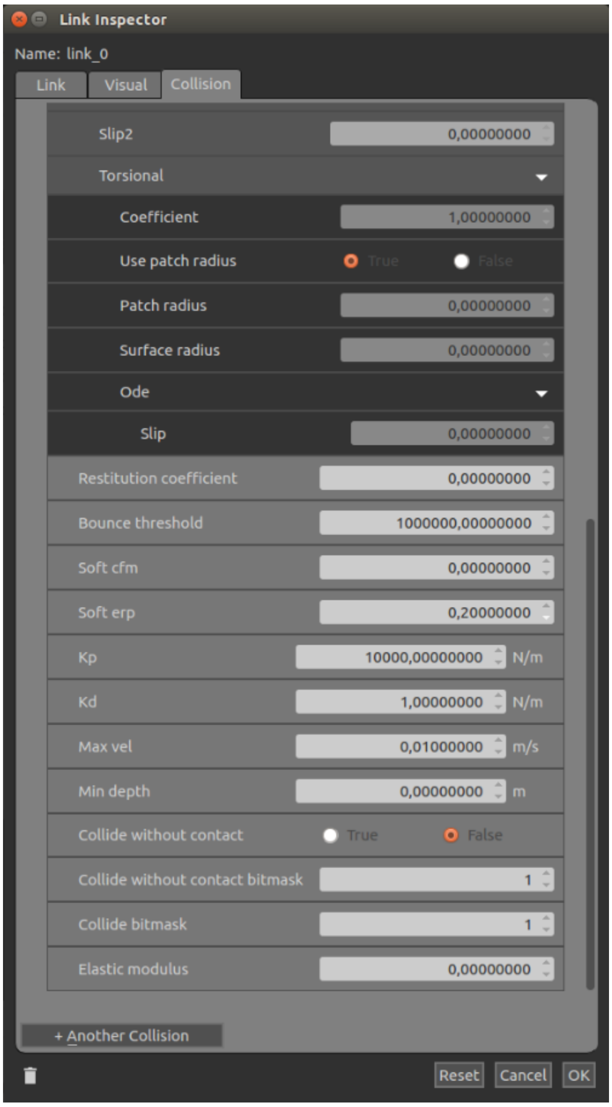
Custom X 0,000000 Y 0,000000 Z 0,000000

Slip1 0,00000000

Slip2 0,00000000

Torsional

Coefficient 1,00000000



Name: link_0

Slip2 0,00000000

Torsional

Coefficient 1,00000000

Use patch radius ● True ○ False

Patch radius 0,00000000

Surface radius 0,00000000

Ode

Slip 0,00000000

Restitution coefficient 0,00000000

Bounce threshold 1000000,00000000

Soft cfm 0,00000000

Soft erp 0,20000000

Kp 10000,00000000 N/m

Kd 1,00000000 N/m

Max vel 0,01000000 m/s

Min depth 0,00000000 m

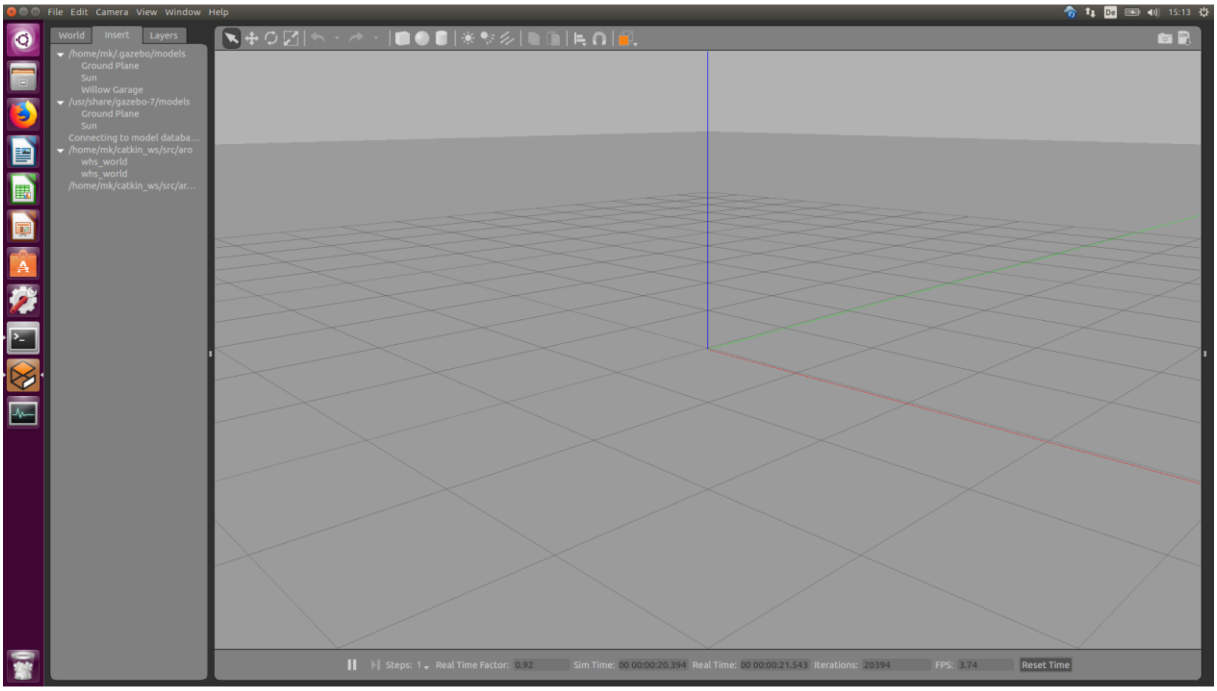
Collide without contact ○ True ● False

Collide without contact bitmask 1

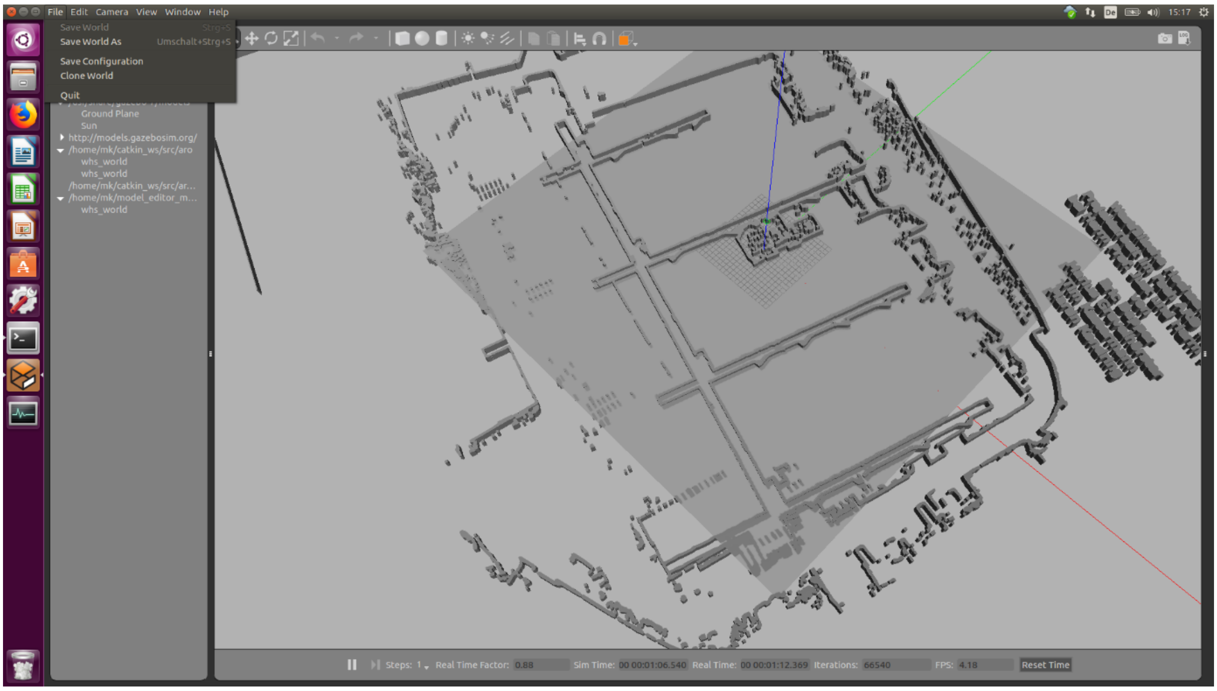
Collide bitmask 1

Elastic modulus 0,00000000

In den Building-Editor wechseln.


File Edit Camera View Window Help
15:13
World Insert Layers
/home/mk/.gazebo/models
Ground Plane
Sun
Willow Garage
/usr/share/gazebo-7/models
Ground Plane
Sun
Connecting to model databa...
/home/mk/catkin_ws/src/aro
whs_world
whs_world
/home/mk/catkin_ws/src/ar...
Steps: 1
Real Time Factor: 0.92
Sim Time: 00 00:00:20.394
Real Time: 00 00:00:21.543
Iterations: 20394
FPS: 3.74
Reset Time

whs_world einfügen und als whs_world.world abspeichern

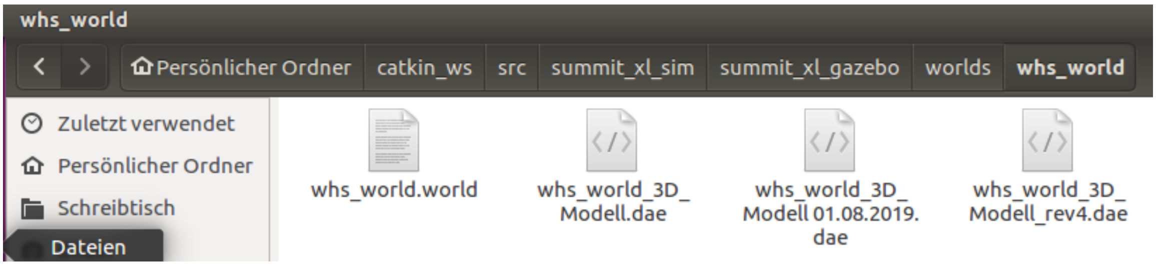

File Edit Camera View Window Help
15:17
Save World Strg+S
Save World As Umschalt+Strg+S
Save Configuration
Clone World
Quit
Ground Plane
Sun
http://models.gazebosim.org/
/home/mk/catkin_ws/src/aro
whs_world
whs_world
/home/mk/catkin_ws/src/ar...
/home/mk/model_editor_m...
whs_world
Steps: 1
Real Time Factor: 0.88
Sim Time: 00 00:01:06.540
Real Time: 00 00:01:12.369
Iterations: 66540
FPS: 4.18
Reset Time

1.5 Anpassen verschiedener Dateien

1.5.1 Für Gazebo:

Nach ~/summit_xl_gazebo/worlds navigieren und Ordner whs_world anlegen

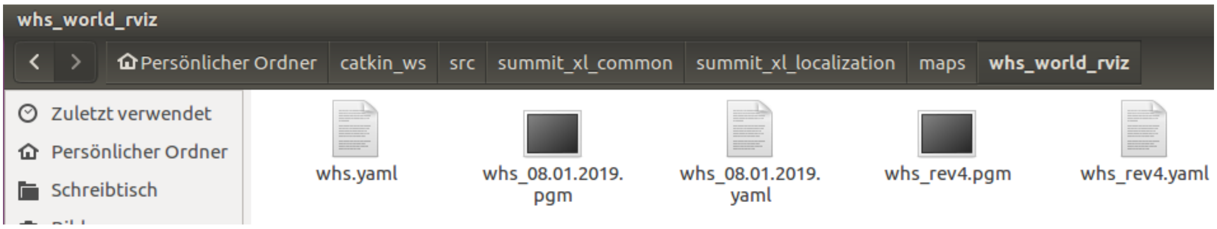
Die erstellte .world und .dae Datei ablegen



1.5.2 Für RVIZ

Nach ~/summit_xl_loclization/maps navigieren und Ordner whs_world_rivz anlegen

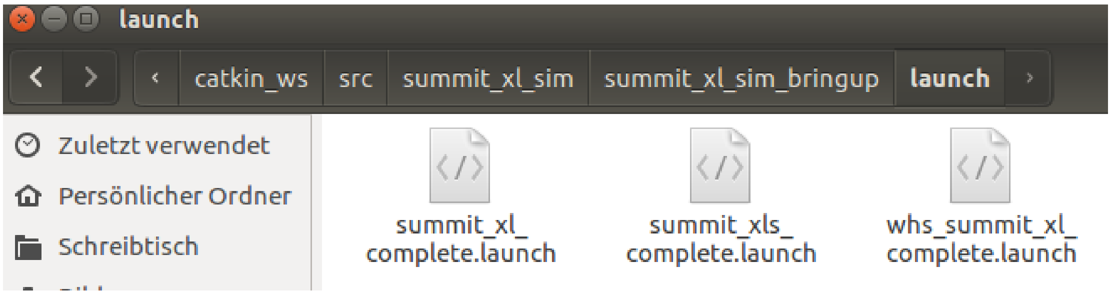
Die mit der Karte erzeuge .yaml unddie .pgm Datei ablegen in whs umbenennen



1.5.3 Starten realer und simulierter Summit

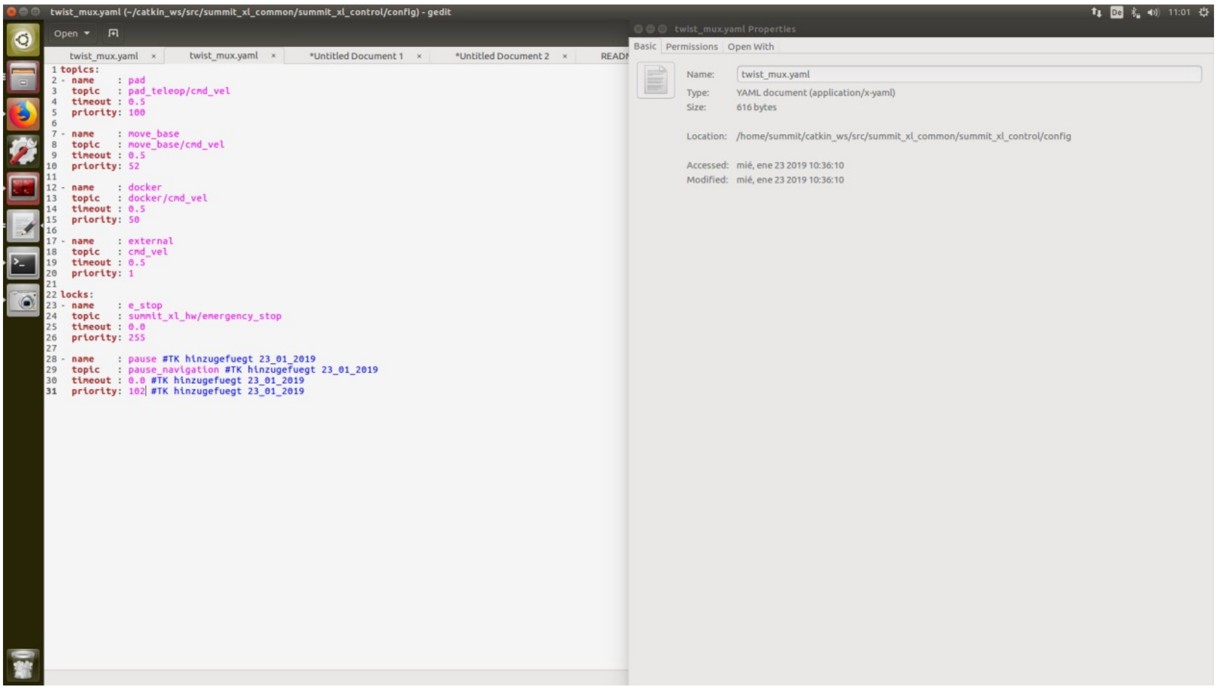
Die whs_summit_xl_complete.launch öffnen

Bei Aufruf der RVIZ und Gazebo Umgebung die angepassten Parameter übergeben



1.5.4 Stoppen des Summit über Twist-Mux

Hierbei muss die Priorität zwischen dem PS4-Pad (100) und der move_base (50) liegen. Hier wurde zu testzwecken die Priorität auf 102 eingestellt, dadurch ist der komplette Summit gesperrt und kann auch nicht über das Pad gefahren werden.



twist_mux.yaml (~/catkin_ws/src/summit_xl_common/summit_xl_control/config) - gedit
Open
twist_mux.yaml
twist_mux.yaml
*Untitled Document 1
*Untitled Document 2
READM
11:01
twist_mux.yaml Properties
Basic
Permissions
Open With

```yaml
topics:
- name     : pad
  topic    : pad_teleop/cmd_vel
  timeout  : 0.5
  priority: 100

- name     : move_base
  topic    : move_base/cmd_vel
  timeout  : 0.5
  priority: 52

- name     : docker
  topic    : docker/cmd_vel
  timeout  : 0.5
  priority: 50

- name     : external
  topic    : cmd_vel
  timeout  : 0.5
  priority: 1

locks:
- name     : e_stop
  topic    : summit_xl_hw/emergency_stop
  timeout  : 0.0
  priority: 255

- name     : pause #TK hinzugefuegt 23_01_2019
  topic    : pause_navigation #TK hinzugefuegt 23_01_2019
  timeout  : 0.0 #TK hinzugefuegt 23_01_2019
  priority: 102 #TK hinzugefuegt 23_01_2019
```

Name: twist_mux.yaml
Type: YAML document (application/x-yaml)
Size: 616 bytes

Location: /home/summit/catkin_ws/src/summit_xl_common/summit_xl_control/config

Accessed: mié, ene 23 2019 10:36:10
Modified: mié, ene 23 2019 10:36:10

1.6 Python-Skript

1.6.1 Gazebo

```
import rospy
import sys
import time
import dynamic_reconfigure.client
import signal
from geometry_msgs.msg import Twist, PoseStamped, Pose, PoseWithCovarianceStamped, PointStamped
from sensor_msgs.msg import LaserScan
from math import pow, atan2, sqrt,sin,cos,pi
from Tkinter import *
from std_msgs.msg import *

global stop_goal

class SetNumberOfPoints(Frame):
    #Initialization of the first Frame
    def __init__(self,master=None):
        Frame.__init__(self, master)
        Button(master,text='Choose Number of Points',width=20,command=self.update_NumberOfPoints).grid(row=3,column=0)    #Button for Number input
        Button(master,text='Number of Points is set',width=20,command=root.destroy).grid(row=3, column=1)
    #Button for frame closing

    #Function to define number of points
    def update_NumberOfPoints(self):
        self.numberOfPoints = input("Set the number of marker points: ")            #Number of points can be selected
        if self.numberOfPoints == 2:
            self.matrix=[[0,0],[0,0]]            #If and elseif defined the matrix
        elif self.numberOfPoints == 3:
            self.matrix=[[0,0],[0,0],[0,0]]
        elif self.numberOfPoints == 4:
            self.matrix=[[0,0],[0,0],[0,0],[0,0]]
        elif self.numberOfPoints == 5:
            self.matrix=[[0,0],[0,0],[0,0],[0,0],[0,0]]
        elif self.numberOfPoints == 6:
            self.matrix=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]
        elif self.numberOfPoints == 7:
            self.matrix=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]
        elif self.numberOfPoints == 8:
            self.matrix=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]
        else:
            self.matrix=[[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]
        return self.matrix,self.numberOfPoints                                          #Matrix and numberOfPoints
are returned

class SetMarkers(Frame):
    #Initialization of the first Frame
    def __init__(self,master=None):
        Frame.__init__(self, master)
        #--- 2 Buttons OK und Abbrechen ---
        Button(master,text='Route is defined',width=20,command=root2.destroy).grid(row=3, column=1)

class summit_xl_a:
    #Initialization of the first Frame
    def __init__(self,matrix,numberOfPoints):
        global stop_goal
        #Initialization of the summit node
        rospy.init_node('summit_xl_a', anonymous=True)
        #Initialization of publisher and subscriber
        self.goal_publisher = rospy.Publisher('/summit_xl_a/move_base_simple/goal', PoseStamped, queue_size=128)
        #published the goals
        self.pause_navigation = rospy.Publisher('/summit_xl_a/pause_navigation', std_msgs.msg.Bool,queue_size=10)
        #published navgation stopped
        self.pose_subscriber = rospy.Subscriber('/summit_xl_a/amcl_pose', PoseWithCovarianceStamped, self.update_pose)
    #subscribes the current position of the summit
        self.point_subscriber = rospy.Subscriber('/clicked_point', PointStamped, self.update_point)
    #subscribes the clicked_point/published_Point
        self.SubScan = rospy.Subscriber('summit_xl_a/front_laser/scan',LaserScan,self.obstacle_check)
    #subscriber the LaserScans
        #Initialization of the message structure
        self.point = PointStamped()
            #message structure of the goal
        self.pose = PoseWithCovarianceStamped()
            #message structure of the amcl position
        self.scan = LaserScan()
                #message structure of the LaserScans
        #define variables for the program
        self.rate = rospy.Rate(10)
            #update intervall
        self.currentMarker=0
            #counter for the marker
        self.goal_msg = PoseStamped()
            #self.goal_msg gets the structure of the PoseStamped
        self.matrix=matrix
            #matrix for the goals
        self.numberOfPoints=numberOfPoints
            #variable of the number of choosen points
        self.Summit_blocked = 0
                #variable, if summit is stopped
        self.blockedRanges = 40
                #number of ranges, which have to be blocked to stop the summit
        self.blockedDistance = 0.9
            #limit of the distance, which should be allowed
        self.scan_number = 0
            #counter for the scans
        self.clientini_TebLocal=            dynamic_reconfigure.client.Client('/summit_xl_a/move_base/TebLocalPlanner-
ROS',timeout=30)            #reconfigure Client to stop the summit
        self.clientini_amcl = dynamic_reconfigure.client.Client('/summit_xl_a/amcl',timeout=30)
        self.goal_msg.header.frame_id ="/summit_xl_a_map"
        #frame_id of the goal message is always the same
        self.goal_msg.pose.orientation.x=0
            #orientation-x is always 0
        self.goal_msg.pose.orientation.y=0
        param={'max_vel_x': 1.2 ,'max_vel_theta': 0.75 ,'max_vel_x_backwards': 0.5}
        self.clientini_TebLocal.update_configuration(param)
        param2={'beam_skip_threshold': 0.2,'recovery_alpha_slow': 0.001,'recovery_alpha_fast': 0.1,'update_min_a':0.1,'up-
date_min_d':0.1}
        self.clientini_amcl.update_configuration(param2)

    def update_pose(self, data):                    #Update current position in x and y
        self.pose = data
        self.xposition = round(self.pose.pose.pose.position.x, 4)
        self.yposition = round(self.pose.pose.pose.position.y, 4)
        self.zorientation = round(self.pose.pose.pose.orientation.z, 4)
        self.worientation = round(self.pose.pose.pose.orientation.w, 4)

    def update_point(self, data):                                   #write clicked point in goal-
matrix
        self.point = data
        self.pointx = round(self.point.point.x,4)
        self.pointy = round(self.point.point.y,4)
        self.matrix[self.currentMarker][0]=self.pointx              #write x-position in the first
column
        self.matrix[self.currentMarker][1]=self.pointy              #write y-position in the second
column
        self.currentMarker=self.currentMarker+1                     #count the currentMarker for
the raw of the matrix
        print(self.matrix)

    def end_route(self):
        self.goal_msg.pose.position.x=self.xposition
        self.goal_msg.pose.position.y=self.yposition
        self.goal_msg.pose.orientation.z=self.zorientation
        self.goal_msg.pose.orientation.w=self.worientation
        self.goal_publisher.publish(self.goal_msg)
        print('end')
        sys.exit(0)

    def euclidean_distance(self, goal_pose):                   #Calculates the distance between the cur-
rent position and the goal
        return sqrt(pow((goal_pose.pose.position.x - self.xposition), 2) + pow((goal_pose.pose.position.y - self.yposition), 2))

    def move2goal(self):                        #Control of the ride
        global stop_goal
        distance_tolerance = 1 #input("Set your tolerance: ")      #set goal radius
        reached_goals=0                                                  #counter for
reached goals
        a=0
        print("Let's move your robot")
        lap=0                                                            #counter for the
laps
        while lap<2:                                                     #first laps will be
counted up in the matrix
            while reached_goals<self.numberOfPoints:               #all goals of the lap will be reached
                self.goal_msg.pose.position.x=self.matrix[reached_goals][0]     #X-position is got out of the
first column
                self.goal_msg.pose.position.y=self.matrix[reached_goals][1]     #Y-position is got out of the
second column
                theta=atan2((self.matrix[reached_goals][1]-self.yposition),(self.matrix[reached_goals][0]-
self.xposition))#theta is calculated as atan2 of diff_y and diff_x between current position and goal
                self.goal_msg.pose.orientation.z=sin(theta/2)               #orientation-z is addicted of
theta
                self.goal_msg.pose.orientation.w=cos(theta/2)               #orientation-w is addicted of
theta
                self.goal_publisher.publish(self.goal_msg)                  #send goal message
                while self.euclidean_distance(self.goal_msg) >= distance_tolerance:     #check distance to
the goal every 0.5 seconds
                    if stop_goal == 1:
                        self.end_route()
                    else:
                        a=a
            print("reached goal",reached_goals)
                reached_goals=reached_goals+1                                    #count reached
goals
            lap=lap+1                                                   #count driven laps
            reached_goals=0                                                  #set reached goals
to 0 for next lap
            print("Lap",lap)
        self.goal_msg.pose.position.x=self.matrix[reached_goals][0]            #drive to the first goal
        self.goal_msg.pose.position.y=self.matrix[reached_goals][1]
        theta=atan2((self.matrix[reached_goals][1]-self.yposition),(self.matrix[reached_goals][0]-self.xposition))
        self.goal_msg.pose.orientation.z=sin(theta/2)
        self.goal_msg.pose.orientation.w=cos(theta/2)
        self.goal_publisher.publish(self.goal_msg)
        while self.euclidean_distance(self.goal_msg) >= distance_tolerance:
            if stop_goal == 1:
                self.end_route()
            else:
                a=a
        print("direction is changed")
        while lap < 4:                                                  #third and fourth
lap the matrix will be read from the last raw to the first raw
            reached_goals=self.numberOfPoints-1
            while reached_goals>=0:
                self.goal_msg.pose.position.x=self.matrix[reached_goals][0]
                self.goal_msg.pose.position.y=self.matrix[reached_goals][1]
                theta=atan2((self.matrix[reached_goals][1]-self.yposition),(self.matrix[reached_goals][0]-
self.xposition))
                self.goal_msg.pose.orientation.z=sin(theta/2)
                self.goal_msg.pose.orientation.w=cos(theta/2)
                self.goal_publisher.publish(self.goal_msg)
                while self.euclidean_distance(self.goal_msg) >= distance_tolerance:
                    if stop_goal == 1:
                        self.end_route()
                    else:
                        a=a
                print("reached goal",reached_goals)
                reached_goals=reached_goals-1
            lap=lap+1
            print("Lap",lap)
        print("finished")

def handler (signum,frame):
    global stop_goal
    print ('Stop prevent')
    stop_goal=1

signal.signal(signal.SIGTSTP, handler)

if __name__ == '__main__':
    try:
        stop_goal = 0
        root=Tk()                                                   #TK for the first frame
        app=SetNumberOfPoints(master=root)                          #link the first class to the first
frame
        app.mainloop()                                                  #loop of the first
frame
        root2=Tk()                                                  #TK for the second frame
        app2=SetMarkers(master=root2)                                   #link the second
class to the second frame
        x = summit_xl_a(app.matrix,app.numberOfPoints)              #Initialization of the summit
class, include matrix and number of points of the first class
        app2.mainloop()                                                 #loop of the second
frame
        x.move2goal()                                                   #control the sum-
mit during the drive

    except rospy.ROSInterruptException:
        pass
```

1.6.2 Summit

```
import rospy
import sys
import time
import dynamic_reconfigure.client
import signal
from geometry_msgs.msg import Twist, PoseStamped, Pose, PoseWithCovarianceStamped, PointStamped
from sensor_msgs.msg import LaserScan
from math import pow, atan2, sqrt,sin,cos,pi
from Tkinter import *
from std_msgs.msg import *
from subprocess import call
```

```
numberOfPoints = 16
matrix =[[-131.6924, -28.2466], [-134.1517, -45.3383], [-129.9457, -56.4002], [-96.9162, -60.335], [-51.2221, -66.2565],[-30, -67], [-10.6616,
-68.9937], [-2.3376, -57.0268], [-1.8059, -11.2001], [1.3191, 10.3079], [-12.1881, 18.2818], [-68.2686, 24.6036], [-95.6797, 30.352], [-
110.3519, 31.9536], [-115.3365, 29.2044], [-120.4358, 5.2322]]


global stop_goal


class SetMarkers(Frame):
        #Initialization of the first Frame
        def __init__(self,master=None):
                Frame.__init__(self, master)
                #--- 2 Buttons OK und Abbrechen ---
                Button(master,text='Route is defined',width=20,command=root2.destroy).grid(row=3, column=1)

class summit_xl:
        #Initialization of the first Frame
        def __init__(self,matrix,numberOfPoints):
                global stop_goal
                #Initialization of the summit node
                print('Initialisation')
                rospy.init_node('summit_xl', anonymous=True)
                #Initialization of publisher and subscriber
                self.goal_publisher = rospy.Publisher('/summit_xl/move_base_simple/goal', PoseStamped, queue_size=128)
        #published the goals
                self.pause_navigation = rospy.Publisher('/summit_xl/pause_navigation', std_msgs.msg.Bool,queue_size=10)
        #published navgation stopped
                self.pose_subscriber  =  rospy.Subscriber('/summit_xl/amcl_pose',  PoseWithCovarianceStamped,  self.update_pose)
        #subscribes the current position of the summit
                self.point_subscriber = rospy.Subscriber('/clicked_point', PointStamped, self.update_point)
        #subscribes the clicked_point/published_Point
                #self.SubScan = rospy.Subscriber('summit_xl/front_laser/scan',LaserScan,self.obstacle_check)
        #subscriber the LaserScans
                #Initialization of the message structure
                self.point = PointStamped()
                        #message structure of the goal
                self.pose = PoseWithCovarianceStamped()
                        #message structure of the amcl position
                self.scan = LaserScan()
                                #message structure of the LaserScans
                #define variables for the program
                self.rate = rospy.Rate(10)
                        #update intervall
                self.currentMarker=0
                        #counter for the marker
                self.goal_msg = PoseStamped()
                        #self.goal_msg gets the structure of the PoseStamped
                self.matrix=matrix
                        #matrix for the goals
                self.numberOfPoints=numberOfPoints
                        #variable of the number of choosen points
                self.Summit_blocked = 0
                                #variable, if summit is stopped
                self.blockedRanges = 40
                                #number of ranges, which have to be blocked to stop the summit
                self.blockedDistance = 0.9
                        #limit of the distance, which should be allowed
                self.scan_number = 0
                        #counter for the scans
                self.clientini_TebLocal=                dynamic_reconfigure.client.Client('/summit_xl/move_base/TebLocalPlanner-
ROS',timeout=30)        #reconfigure Client to stop the summit
                #self.clientini_amcl = dynamic_reconfigure.client.Client('/summit_xl/amcl',timeout=30)
                self.goal_msg.header.frame_id ="/summit_xl_map"
                #frame_id of the goal message is always the same
                self.goal_msg.pose.orientation.x=0
                #orientation-x is always 0
                self.goal_msg.pose.orientation.y=0
                param={'max_vel_x': 3.0 ,'max_vel_theta': 0.75 ,'max_vel_x_backwards': 0.5}
                self.clientini_TebLocal.update_configuration(param)
                #param2={'beam_skip_threshold': 0.3,'recovery_alpha_slow': 0.001,'recovery_alpha_fast': 0.1,'update_min_a':0.1,'update_min_d':0.1}
                #param2={'recovery_alpha_slow': 0.001,'recovery_alpha_fast': 0.1}
                #self.clientini_amcl.update_configuration(param2)
                call(["rosservice","call","/summit_xl/move_base/clear_costmaps"])

        def update_pose(self, data):                    #Update current position in x and y
                self.pose = data
                self.xposition = round(self.pose.pose.pose.position.x, 4)
                self.yposition = round(self.pose.pose.pose.position.y, 4)
                self.zorientation = round(self.pose.pose.pose.orientation.z, 4)
                self.worientation = round(self.pose.pose.pose.orientation.w, 4)

        def update_point(self, data):                                           #write clicked point in goal-matrix
                self.point = data
                self.pointx = round(self.point.point.x,4)
                self.pointy = round(self.point.point.y,4)
                self.matrix[self.currentMarker][0]=self.pointx                      #write x-position in the first column
                self.matrix[self.currentMarker][1]=self.pointy                      #write y-position in the second column
                self.currentMarker=self.currentMarker+1                             #count the currentMarker for the raw of the matrix
                print(self.matrix)

        def end_route(self):
                self.goal_msg.pose.position.x=self.xposition
                self.goal_msg.pose.position.y=self.yposition
                self.goal_msg.pose.orientation.z=self.zorientation
                self.goal_msg.pose.orientation.w=self.worientation
                self.goal_publisher.publish(self.goal_msg)
                print('end')
                sys.exit(0)

        def euclidean_distance(self, goal_pose):                          #Calculates the distance between the current position and the goal
                return sqrt(pow((goal_pose.pose.position.x - self.xposition), 2) + pow((goal_pose.pose.position.y - self.yposition), 2))

        def move2goal(self):                            #Control of the ride
                global stop_goal
                distance_tolerance = 1 #input("Set your tolerance: ")         #set goal radius
                reached_goals=0                                                     #counter for reached goals
                a=0
                print("Let's move your robot")
                lap=0                                                               #counter for the laps
                while lap<2:                                                        #first laps will be counted up in the matrix
                        while reached_goals<self.numberOfPoints:                  #all goals of the lap will be reached
                                self.goal_msg.pose.position.x=self.matrix[reached_goals][0]     #X-position is got out of the first column
                                self.goal_msg.pose.position.y=self.matrix[reached_goals][1]     #Y-position is got out of the second column
                                theta=atan2((self.matrix[reached_goals][1]-self.yposition),(self.matrix[reached_goals][0]-self.xposition))#theta is calculated as atan2 of diff_y and diff_x between current position and goal
                                self.goal_msg.pose.orientation.z=sin(theta/2)                   #orientation-z is addicted of theta
                                self.goal_msg.pose.orientation.w=cos(theta/2)                   #orientation-w is addicted of theta
                                self.goal_publisher.publish(self.goal_msg)                      #send goal message
                                while self.euclidean_distance(self.goal_msg) >= distance_tolerance:      #check distance to the goal every 0.5 seconds
                                    if stop_goal == 1:
                                        self.end_route()
                                    elif sqrt(pow((self.xposition-self.matrix[reached_goals-1][0]),2)+pow((self.yposition-self.matrix[reached_goals-1][1]),2)) <= distance_tolerance:
                                        self.goal_publisher.publish(self.goal_msg)                      #send goal message again
                                        print('resend goal')
                                        time.sleep(8)
                        print("reached goal",reached_goals)
                                reached_goals=reached_goals+1                    #count reached goals
                                call(["rosservice","call","/summit_xl/move_base/clear_costmaps"])
                        lap=lap+1                                            #count driven laps
                        reached_goals=0                                      #set reached goals to 0 for next lap
                        print("Lap",lap)
                self.goal_msg.pose.position.x=self.matrix[0][0]          #drive to the first goal
                self.goal_msg.pose.position.y=self.matrix[0][1]
                theta=atan2((self.matrix[0][1]-self.yposition),(self.matrix[0][0]-self.xposition))
                self.goal_msg.pose.orientation.z=sin(theta/2)
                self.goal_msg.pose.orientation.w=cos(theta/2)
                self.goal_publisher.publish(self.goal_msg)
                while self.euclidean_distance(self.goal_msg) >= distance_tolerance:
                        if stop_goal == 1:
                                self.end_route()
                        elif sqrt(pow((self.xposition-self.matrix[self.numberOfPoints-1][0]),2)+pow((self.yposition-self.matrix[numberOfPoints-1][1]),2)) <= distance_tolerance:
                                self.goal_publisher.publish(self.goal_msg)       #send goal message again
                                print('resend goal')
                                time.sleep(8)
                print("direction is changed")
                call(["rosservice","call","/summit_xl/move_base/clear_costmaps"])
                while lap < 4:                                           #third and fourth lap the matrix will be read from the last raw to the first raw
                        reached_goals=self.numberOfPoints-1
                        while reached_goals>=0:
                                self.goal_msg.pose.position.x=self.matrix[reached_goals][0]
                                self.goal_msg.pose.position.y=self.matrix[reached_goals][1]
                                theta=atan2((self.matrix[reached_goals][1]-self.yposition),(self.matrix[reached_goals][0]-self.xposition))
                                self.goal_msg.pose.orientation.z=sin(theta/2)
                                self.goal_msg.pose.orientation.w=cos(theta/2)
                                self.goal_publisher.publish(self.goal_msg)
                                while self.euclidean_distance(self.goal_msg) >= distance_tolerance:
                                    if stop_goal == 1:
                                        self.end_route()
                                    if reached_goals==self.numberOfPoints-1:
                                        if sqrt(pow((self.xposition-self.matrix[0][0]),2)+pow((self.yposition-self.matrix[0][1]),2)) <= distance_tolerance:
                                            self.goal_publisher.publish(self.goal_msg)
                                            print('resend goal')
                                            time.sleep(8)
                                    elif sqrt(pow((self.xposition-self.matrix[reached_goals+1][0]),2)+pow((self.yposition-self.matrix[reached_goals+1][1]),2)) <= distance_tolerance:
                                        self.goal_publisher.publish(self.goal_msg)       #send goal message again
                                        print('resend goal')
                                        time.sleep(8)
                                print("reached goal",reached_goals)
                                reached_goals=reached_goals-1
                                call(["rosservice","call","/summit_xl/move_base/clear_costmaps"])
                        lap=lap+1
                        print("Lap",lap)
                print("finished")

def handler (signum,frame):
    global stop_goal
    print ('Stop prevent')
    stop_goal=1

signal.signal(signal.SIGTSTP, handler)

if __name__ == '__main__':
    try:
        stop_goal = 0
        root2=Tk()                                               #TK for the second frame
        app2=SetMarkers(master=root2)                            #link the second class to the second frame
        x = summit_xl(matrix,numberOfPoints)                     #Initialization of the summit class, include matrix and number of points of the first class
```

```
			app2.mainloop()								#loop of the second
frame
			x.move2goal()								#control the sum-
mit during the drive

		except rospy.ROSInterruptException:
			pass
```